Razonamiento y planificación automática

Nerea Luis Mingueza / Alejandro Cervantes

Tema 5: Búsqueda informada (II)

Índice de la clase

Tema 5

Búsqueda A*

Búsqueda por subobjetivos

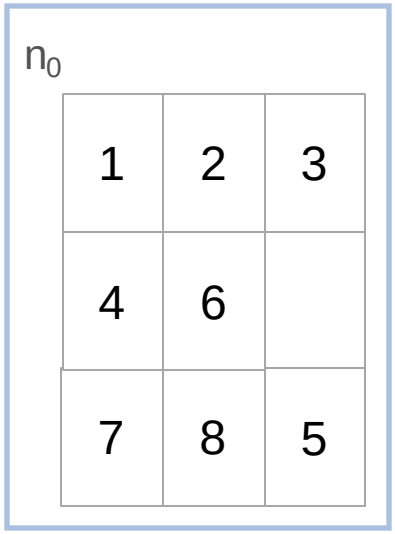
Búsqueda online

Heurísticas

Definición de heurística

Función heurística : Función h(n) que permite estimar el coste desde un determinado estado hasta un estado meta (solución) por el mejor camino posible. Su valor está asociado al **problema**, al **estado (n)** y a **la(s) meta(s)**.

Distintas metas, distinto valor de h



META

1	2	3
4	5	6
7	8	

1	2	3
4		5
8	7	6

Ambas son admisibles (siempre dan un número menor o igual que el verdadero coste de llegar a la meta)

$h_1(n)$ = número de casillas descolocadas

$h_2(n)$ = suma de distancias de las casillas a su destino

$h_1(n_0)=2$

$h_2(n_0)=3$

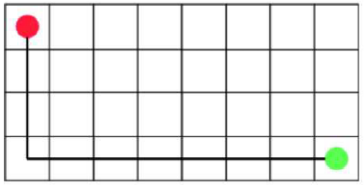
$h_1(n_0)=4$

$h_2(n_0)=5$

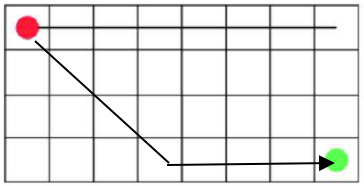
h_2 siempre da valores mayores que h_1. Esto quiere decir que h_2 es más informada (y en general será más útil)

Heurísticas ‘famosas’en el plano

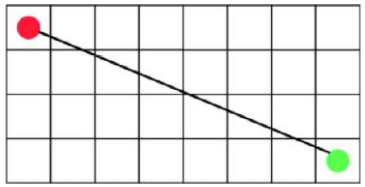
Manhattan Distance:
h = abs (current.x – goal.x) + abs (current.y – goal.y)



Diagonal Distance:
h = max { abs(current.x – goal.x), abs(current.y – goal.y) }



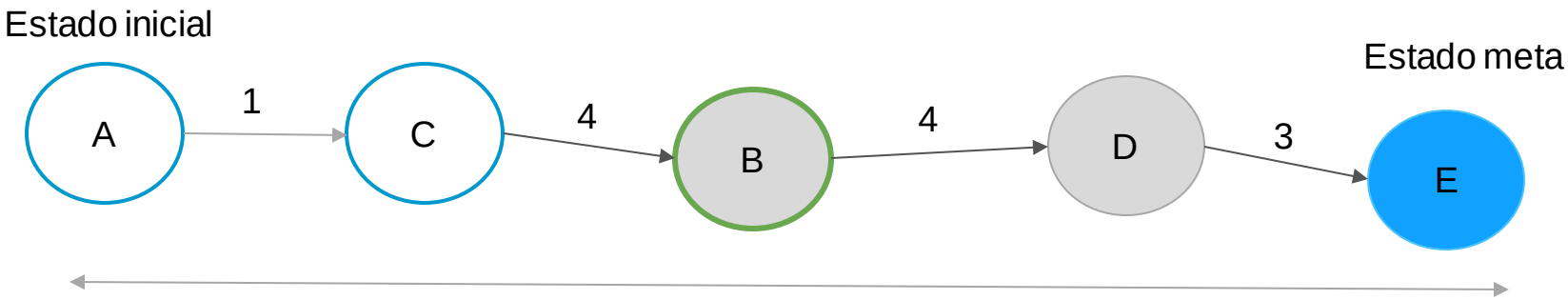
Euclidean Distance:
h = sqrt ((current.x – goal.x)2 + (current.y – goal.y)2)



Importante: una cosa son las acciones permitidas por el problema, y otra independiente la función heurística

Heurística admisible

Definiremos una función heurística *h** como **admisible** si y sólo si $h^*(n) \leq h_{opt}(n)$ para todo *n* (siendo h_{opt} (n) el coste real de llegar a una meta por el mejor camino)



$h_{opt}(B) = 7$
$h^*(B) \leq 7$

h_{opt}(n) coste real de alcanzar el objetivo por el mejor camino posible (desconocido)
h*(n) heurística admisible, coste estimado (a priori) de alcanzar el objetivo.

Una **heurística consistente** es aquella que cumple la propiedad de que el valor de la función de evaluación $f^*(n) = g(n) + h^*(n)$ de un hijo nunca es menor que el del padre.

→ Una heurística consistente siempre es admisible.

Creación de heurísticas mediante relajación de restricciones

Dado un problema real donde tenemos un estado inicial, estado meta, y operadores o acciones, cada una con sus precondiciones (elementos que se deben cumplir en un estado o nodo n, para que la acción se pueda ejecutar sobre dicho nodo n generando un nodo n') y efectos (conjunto de elementos que se deben modificar en el nodo n como resultado de la ejecución del operador o acción), ***relajamos los efectos de cada operador eliminando alguna restricción****.*

Restricciones:

- No se puede mover a una casilla ocupada
- El movimiento sólo se puede hacer a casillas adyacentes horizontal o verticamente
- En cada paso, se intercambian los contenidos de dos casillas

Relajaciones:

- Si quitamos las dos primeras restricciones, generamos la heurística de "número de casillas mal colocadas". Cada casilla se ordena con un movimiento.
- Si quitamos la primera restricción, generamos la heurística de la "suma de distancias de Manhattan". Cada casilla se ordena con tantos movimientos como distancia.

Búsqueda A*

Búsqueda A* (a-estrella)

(Hart and Nilsson,1968)

Usa el Algoritmo Genérico con las siguientes peculiaridades:

- En cada nodo calculamos la suma de los costes de las acciones que me han llevado al mismo, g(n)
- Para cada nodo tenemos el valor de **h*** que corresponde al estado, h*(n). Para que A* sea óptimo debe ser admisible (por eso escribimos h*)
- Los nuevos sucesores se añaden ordenados según la función de evaluación:

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

- Se insertan en la lista Abierta nodos si y solo si: el nuevo nodo tiene menor f que todos los competidores con el mismo estado que puedan estar en Abierta o Cerrada (se pueden reabrir nodos cerrados)
- La lista abierta funcionará como cola de prioridad ordenada según f(n).
- Adicionalmente, evita ciclos generales (no escribe nodos que se han generado previamente en el camino al nodo que se expande).
- Termina en cuanto toca expandir un nodo meta

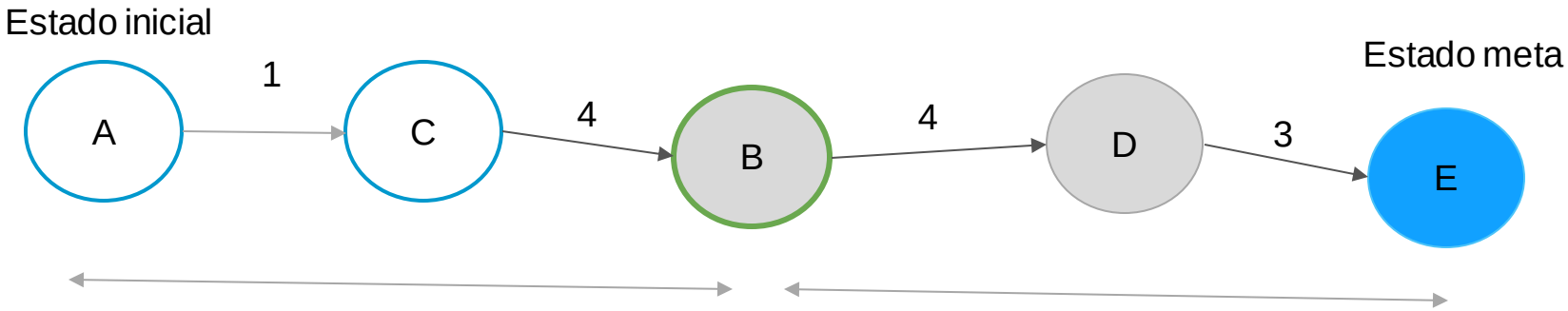
Si h* es admisible, es óptimo en coste

Complejidad exponencial en espacio y tiempo

El algoritmo A*

Dos listas:

- Abierta - cola de prioridades por f(n)
- Cerrada - Nodos visitados (expandidos)



Coste real: g(B) = 1 + 4 = 5

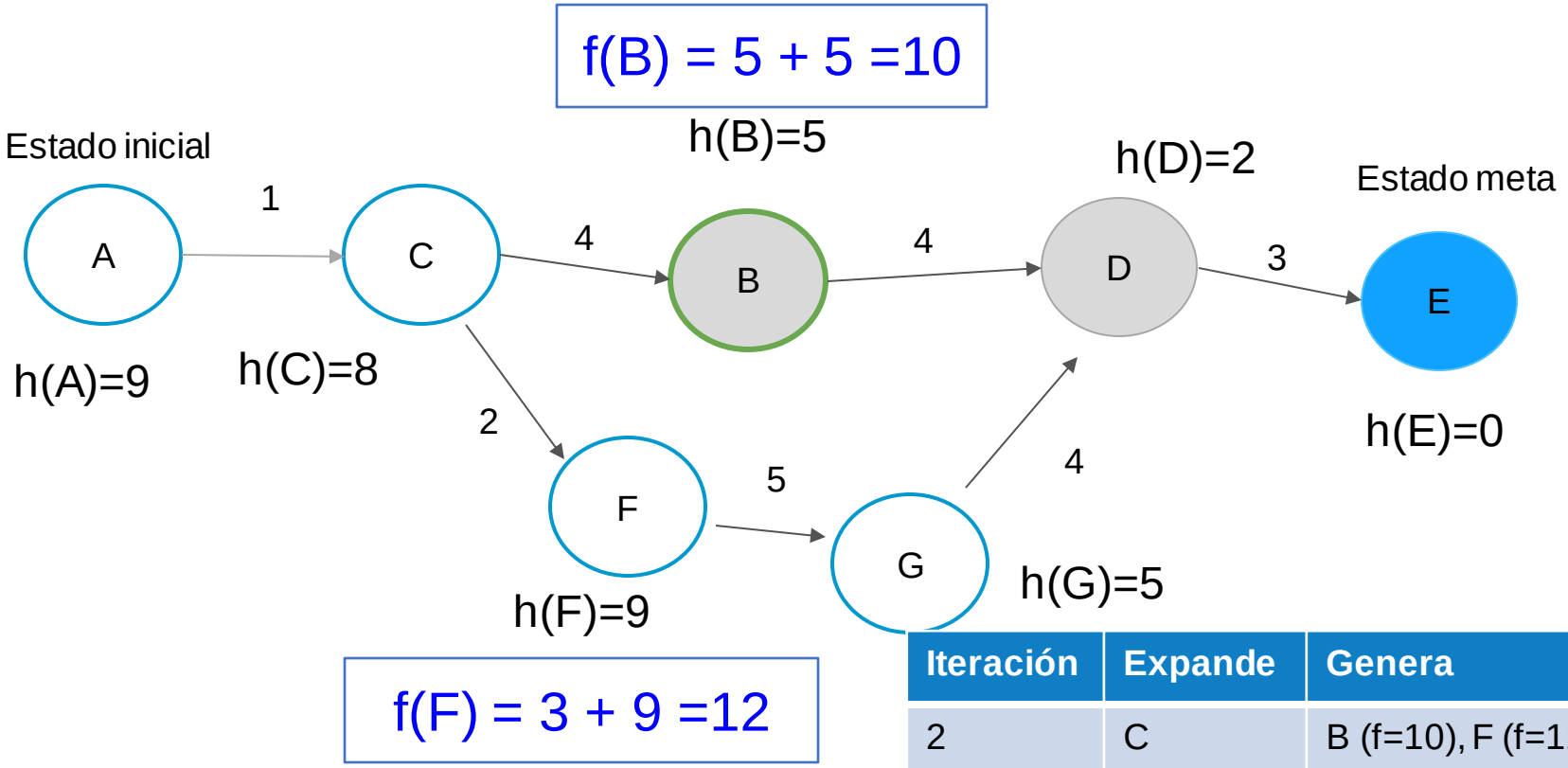
Coste estimado: h*(B).
Si h_{opt} (B) = 7, h*(B) deberá ser 7 o menos

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

f(n): Función de evaluación para la lista abierta
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

El algoritmo A*

Nota: h se habrá definido examinando los operadores y eliminando restricciones

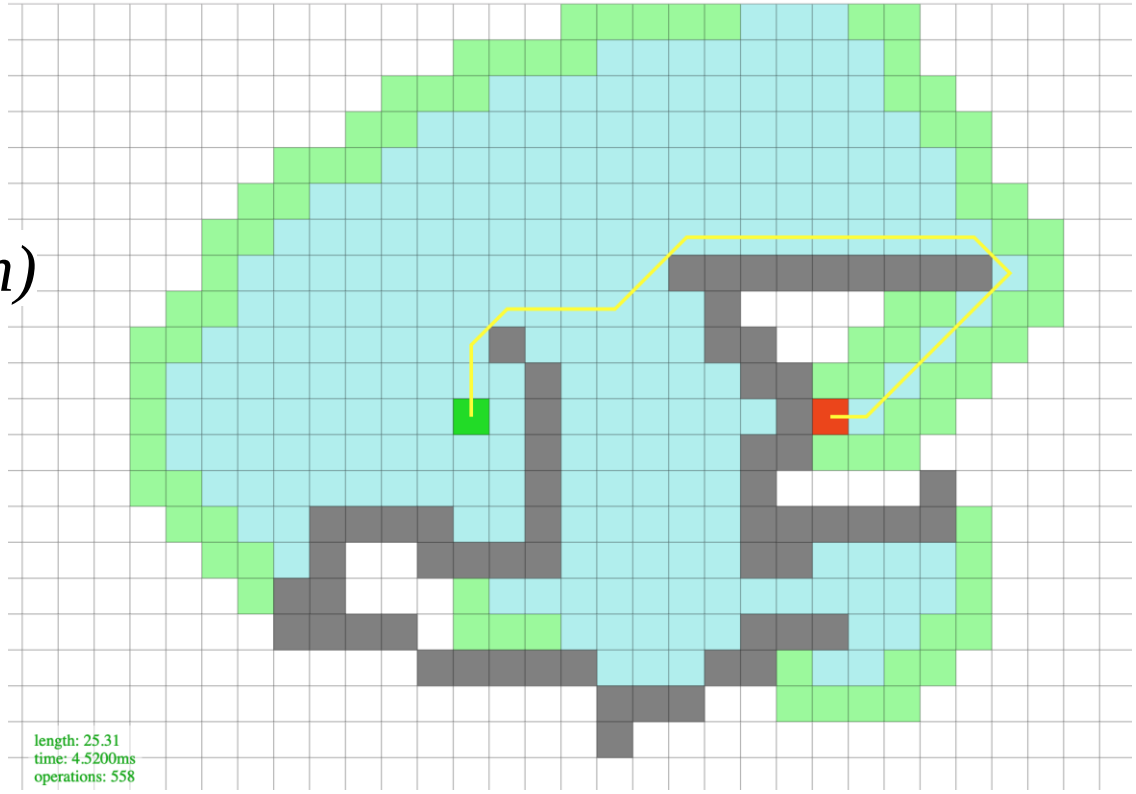


Iteración	Expande	Genera
2	C	B (f=10), F (f=12)

UCS optaría por el nodo F porque el coste de F es g(F)=3, mejor que el de B (g(B)=5). A* tiene una estimación adicional de lo que queda, y al añadirla a la función, en este caso opta por B. Nota: luego podría pasar que F fuese mejor que D y tantear esa opción.
Al final ambos llegarán al mejor camino (ambos son óptimos) pero A* se ahorra pasos

Ejemplo

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



https://qiao.github.io/PathFinding.js/visual/

Propiedades de A*

Completitud: si existe solución, la encuentra.

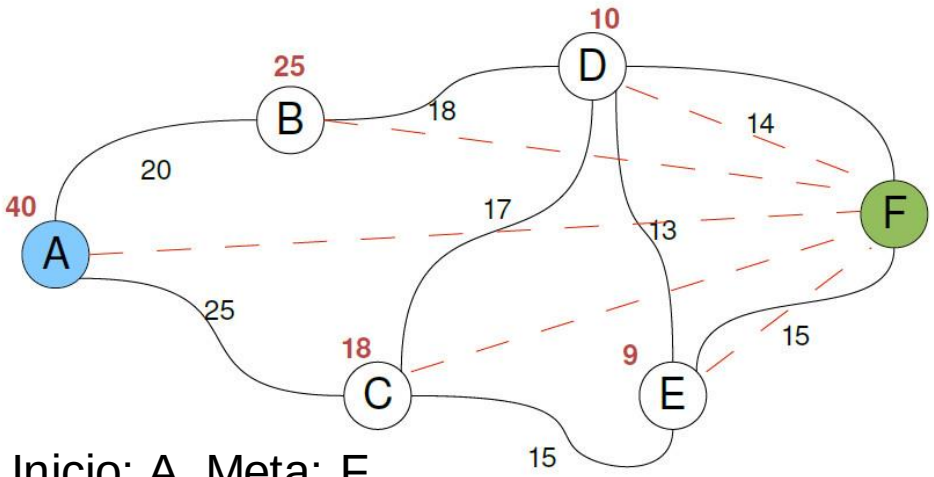
Admisibilidad (optimalidad): si hay una solución óptima, la encuentra siempre I cuando:

- El número de sucesores es finito para cada nodo.
- La función h(n) es admisible: $h^*(n) \leq h_{opt}(n)\ \forall n$

Información: en general si tenemos dos heurísticas admisibles, será preferible usar la más informada (más próxima al valor de $h_{opt}(n)$ pero sin superarlo)

Coste computacional: exponencial en tiempo y espacio. No obstante, si h* es una buena estimación, no expandirá tantos nodos como UCS (Dijkstra)

Ejercicio: Búsqueda en A*

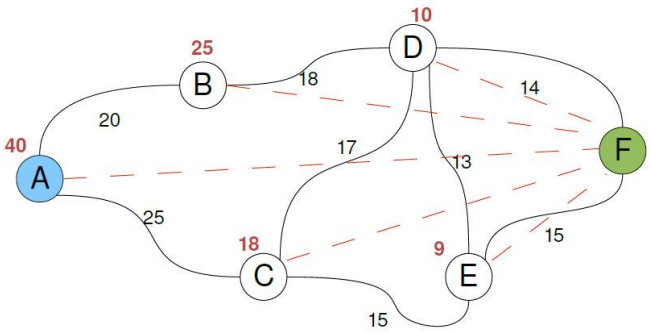


Inicio: A, Meta: F

En negro: coste de las acciones

En rojo: valor de h(n) (¿es admisible?)

Búsqueda en A*



Si no hay nodo con el mismo estado en Abierta, o si el que hay es peor, se reemplaza por el nuevo.

Se comprueba también la lista cerrada y se reabre sólo si el nuevo nodo es mejor.

Expandido	Genera	Abierta	Cerrada
A	B (g=20,h=25,f=45), C(g=25,h=18,f=43)	**B (AB,f=45), C (AC,f=43)**	**A (f=40)**
B (f=45)	D (g=20+18=38, h=10,f=48)	C (AC,f=43), **D (ABD, f=48)**	A (f=40), **B(f=45)**
C (f=43)	D (g=25+17=42, h=10,f=52) (peor) E (g=25+15=40, h=9, f=49)	D (ABD,f=48), **E (ACE,f=49)**	A (f=40), B(f=45), **C (f=43)**
D (f=48)	C (g=38+17=55, h=18, f=73) (peor que el cerrado) E (g=38+13=51, h=9, f=60) (peor) F (g=38+14=52, h=0, f=51)	E (ACE,f=49), **F (ABDF,f=52)**	A (f=40), B(f=45), C (f=43), **D(f=48)**
E (f=49)	C (g=40+15=55, h=18, f= 73) (peor que el cerrado) F (g=40+15=55, h=0,f=53) (peor)	F (ABDF,f=52)	A (f=40), B(f=45), C (f=43), D(f=48), **E(f=49)**
** F (f=51)	FIN		

Solución: A*B(20),BD(18),DF(14).* Hemos anotado el camino a cada nodo en la lista abierta, así es más claro por dónde se llega cuando hay varias opciones.
Coste: 20+18+14=52, **Expandidos**: 5, **Generados**: 10

Resumen Búsqueda

ALGORITMO GENERAL: sacar el primer nodo de la lista abierta; si es meta, termina; si no, expandir (generar sus sucesores), insertar en la lista abierta. NOTA: hay que tener en cuenta que no se generan ciertos nodos repetidos según el algoritmo

Amplitud: los nuevos nodos se introducen al final de ABIERTA (cola, FIFO)

Profundidad: los nuevos nodos se introducen al principio de ABIERTA (pila, LIFO)

Con costes

UCS: la lista ABIERTA se ordena por COSTE TOTAL (g(n), desde el inicio hasta el nodo) (cola con prioridad)

A*: la lista ABIERTA se ordena por COSTE TOTAL (g(n) desde el inicio hasta el nodo) + HEURÍSTICA (h(n)) (cola con prioridad). Para ser óptimo, h(n) debería ser admisible.

Búsqueda por subobjetivos

Búsqueda por subobjetivos

Búsqueda A*: "recorta" partes del árbol que no hace falta evaluar, porque como mínimo "cuesta" h(n) llegar a la meta. Si ya tenemos un camino mejor, no hace falta explorar un camino de elevado f(n)=g(n)+h*(n). Con heurística admisible, encuentra camino óptimo en coste.

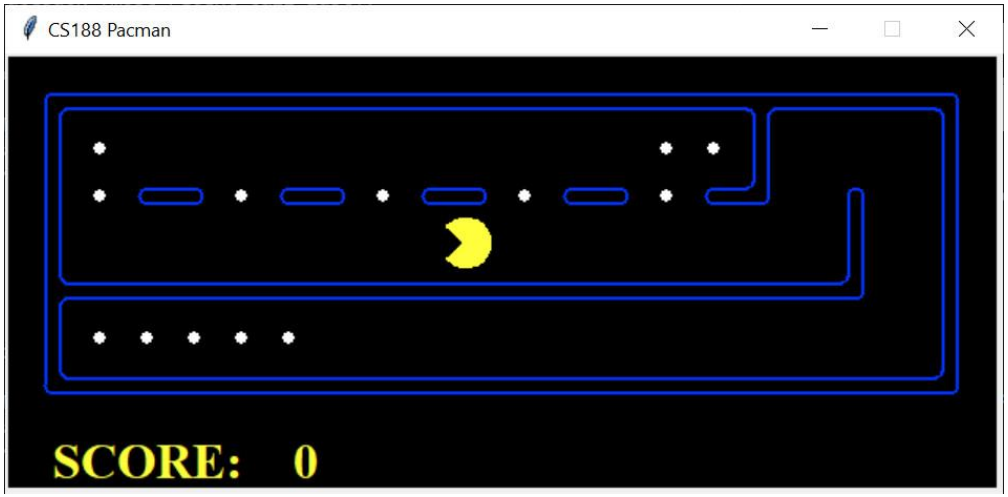
A pesar de la mejora, la complejidad sigue siendo exponencial, es decir, los problemas son intratables en general

Subobjetivos: "descompone" el problema original y aplica algoritmos de búsqueda a cada subproblema. Reduce la complejidad computacional. Pierde la característica de optimalidad (no siempre encontrará el óptimo).

Búsqueda por subobjetivos

Comer B bolas con Amplitud: Meta: situación donde no haya ninguna bola restante. ¿cuántos caminos explora Amplitud?

Comer B bolas con subobjetivos: primero busca la bola más cercana (con Amplitud, por ejemplo y meta: "cualquier bola"); luego repite B veces



Fuente: http://ai.berkeley.edu/search.html

Búsqueda online

Búsqueda online

En cada paso, considera las opciones posibles y escoge la mejor.

Búsqueda online	Búsqueda offline
Repetir: 1. Percibir entorno. 2. Elegir la «mejor» acción. 3. Ejecutar acción. Fin repetir.	1. Percibir entorno. 2. Buscar plan. 3. Ejecutar plan.

Utilidad de la búsqueda online

En algunos entornos:

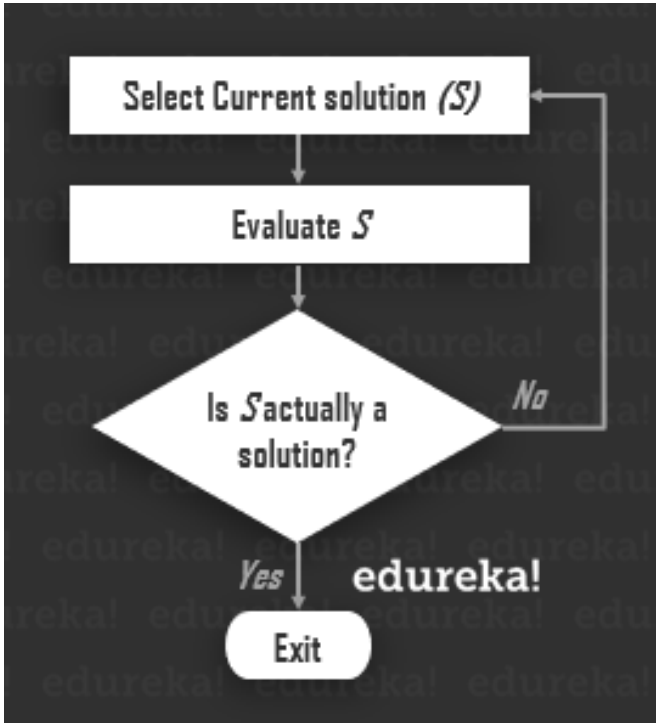
- No es posible/útil encontrar un plan completo. Por ejemplo, en entornos reactivos, o en entornos no deterministas
- No es posible encontrar un plan completo en entornos (parcialmente) inaccesibles, donde se desconoce parte del entorno (estados existentes, resultados de las acciones). Por ejemplo: el dominio de un laberinto donde se conoce parcialmente el laberinto y el mismo se va descubriendo a medida que el agente camina por el laberinto.

Surgen de la necesidad que tienen los agentes reactivos de encontrar una solución en un tiempo muy corto.

Ante falta de tiempo → encontrar una buena acción, no un plan de acción completo bueno.

La idea básica es combinar un paso de búsqueda con un paso de actuación.

Búsqueda online en Escalada



https://www.youtube.com/watch?v=_ThdIOA9Lbk

1. Evaluar el estado actual
2. Si estado actual no es estado meta
   a. Expandir hijos
   b. Moverse al estado que minimice la **heurística** (podría ser otra función y podría ser maximizar)

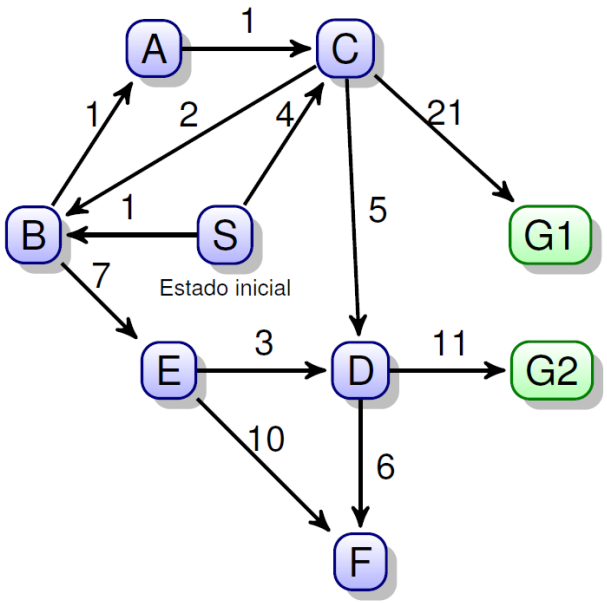
Autoevaluación

- Conocer el concepto de heurística
- Conocer la diferencia entre heurística admisible o no
- Conocer cómo funciona A* y saber simularlo
- Entender el concepto de búsqueda online y poder simular Escalada
- Conocer cómo se calcula la distancia de Manhatta, Diagonal y Euclídea

Soluciones búsqueda no informada

Ejercicio Búsqueda en Amplitud

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos.
Al expandir, el orden es alfabético



Al generar nodos seguimos el orden alfabético indicado

Expandido	Genera	Abierta
S	B, C	B, C
B	A, E	C, A, E
C	D, G1 (anotamos que llegamos a G1 desde C)	A, E, D, G1
A	(C no se genera por repetido)	E, D, G1
E	F (D no se genera por repetido)	D, G1, F
D	G2 (F no se genera por repetido)	G1, F, G2
** G1	FIN	

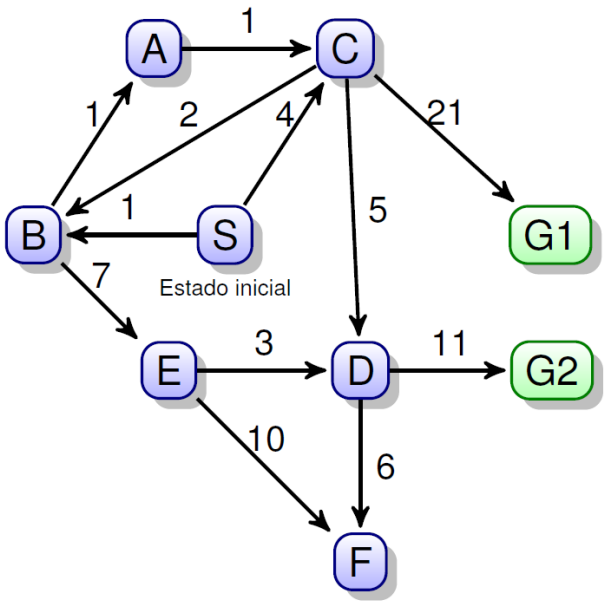
1 Iteraciones 7

No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

Solución: *SC,CG1 .* Tenemos que retrazar a mano el camino porque no pintamos los arcos en la tabla, en el código se irían guardando. Se llega al objetivo **G1**.
Longitud solución: 2, **Expandidos**: 6 (columna 1), **Generados**: 8 (columna 2)

Ejercicio Búsqueda en Profundidad

S: estado inicial, G1 y G2 metas, ignorar los costes en los arcos.
Al expandir, asumamos orden inverso al alfabético



Al generar nodos seguimos el orden indicado

Se añade a la lista abierta por delante (los más recientes se escogen antes)

Expandido	Genera (Sucesores)	Abierta (va acumulando)
S	C, B	**B**, **C**
B	E, A	**A**, **E** , C
A	(C no se genera por repetido)	E, C
E	F, D	**D**, **F**, C
D	G2 (F no se genera por repetido)	**G2**, F, C
** G2	FIN	

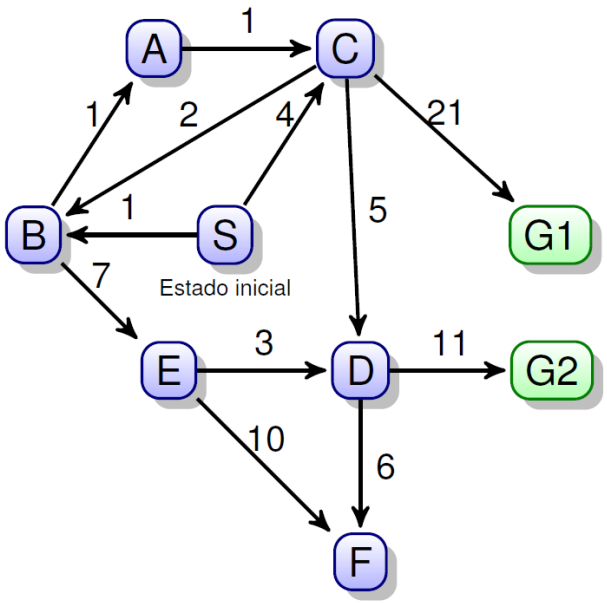
1 Iteraciones 6

No terminamos hasta que no corresponde expandir un nodo objetivo (aunque lo tengamos en la lista Abierta). No generamos ningún nodo con un estado repetido (que esté o haya estado en Abierta)

Solución: *SB, BE, ED, DG2 .* Tenemos que retrazar a mano el camino porque no pintamos los arcos en la tabla, en el código se irían guardando. Se llega al objetivo **G2**.
Longitud solución: 4**, Expandidos**: 5 (columna 1), **Generados**: 7 (columna 2)

Búsqueda con coste uniforme (UCS)

S: estado inicial, G1 y G2 metas, Costes en los arcos. Al expandir, el orden es alfabético, y también se usa este orden para empates en Abierta (a igualdad de g)



Expandido	Genera	Abierta	Cerrada
S	B (g=1), C(g=4)	**B (g=1)**, **C(g=4)**	**S**
B (g=1)	A (g=1+1=2), E (g=1+7=8)	**A (g=2)**, C(g=4), **E(g=8)**	**S,** B (g=1)
A (g=2)	C (g=2+1=3)	**C(g=3)**, ~~**C(g=4),**~~ E(g=8)	**S,** B (g=1), A(g=2)
C (g=3)	D (g=3+5=8), G1 (g=3+21=24) (B repetido en el camino a C)	**D(g=8)**, E(g=8), **G1 (g=24)**	**S,** B (g=1), A(g=2), C(g=3)
D (g=8)	F (g=8+6=14), G2 (g=8+11=19)	E(g=8), **F(g=14)**, **G2 (g=19)**, G1 (g=24)	**S,** B (g=1), A(g=2), C(g=3), D(g=8)
E (g=8)	D (g=8+3=11) (peor que el de la lista cerrada), F (8+10=18) (peor)	F(g=14), G2 (g=19), G1 (g=24)	**S,** B (g=1), A(g=2), C(g=3), D(g=8),E(g=8)
F (g=14)	No tiene sucesores	G2 (g=19), G1 (g=24)	**S,** B (g=1), A(g=2), C(g=3), D(g=8),E(g=8),F(g=14)
** G2 (g=19)	FIN		

1

8

Solución: *SB(1),BA(1),AC(1),CD(5),DG2(11)* . Tenemos que retrazar a mano el camino de coste 19 porque no pintamos los arcos en la tabla, en el código se irían guardando.
Coste: 1+1+1+5+11=19, **Expandidos**: 7, **Generados**: 11 (aunque algunos nunca se insertaron en la lista abierta)

Apéndice: heurísticas para grids

Heurísticas para mapas “de rejilla”

Obtención: Eliminamos la restricción de "no poder traspasar paredes".

Queda la restricción que supone que los movimientos no pueden ser directos, así que calculamos la distancia mediante estimación del número de movimientos.

Este cálculo depende del tipo de operaciones permitidas en el problema. No es lo mismo que se permitan los movimientos diagonales, o que no se permitan.

Cuando las operaciones tienen coste, podemos "escalar" la función heurística en función del coste de los movimientos.

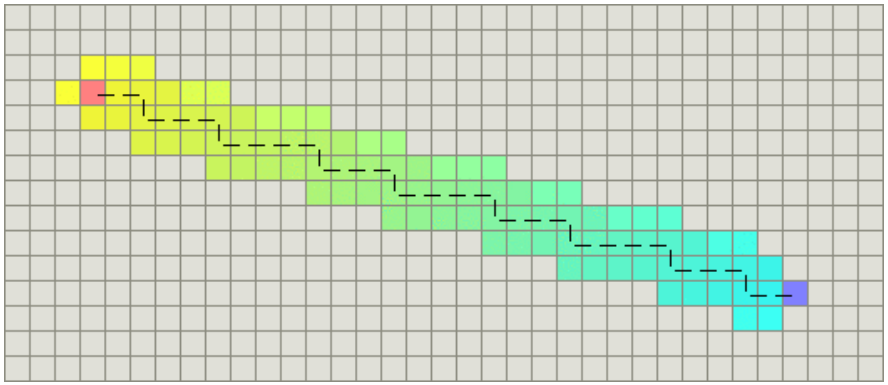
http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#a-stars-use-of-the-heuristic

Heurísticas para mapas “de rejilla”

Manhattan

Sólo movimientos horizontales y verticales

```
function heuristic(node):
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy)
```



D = coste del movimiento

(node.x, node.y)

(goal.x, goal.y)

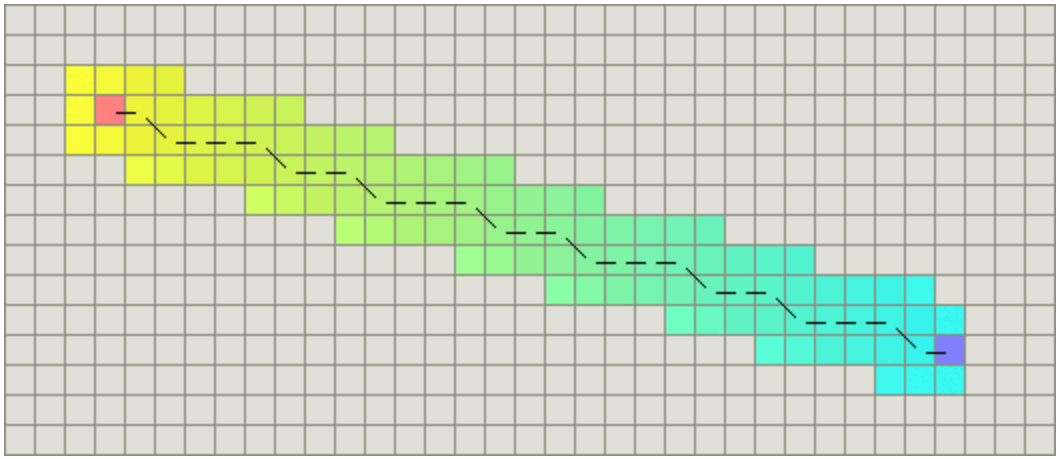
Si sabemos que el coste del movimiento es (como poco) D, podemos multiplicar el valor por este factor sin perder la admisibilidad (y la heurística estará más informada)

Heurísticas para mapas “de rejilla”

Con movimientos diagonales

Diagonal optimizada y con ajuste de costes

```
function heuristic(node):
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
```



D = coste del movimiento v o h
D2 = coste de mover en diagonal

(node.x, node.y)

(goal.x, goal.y)

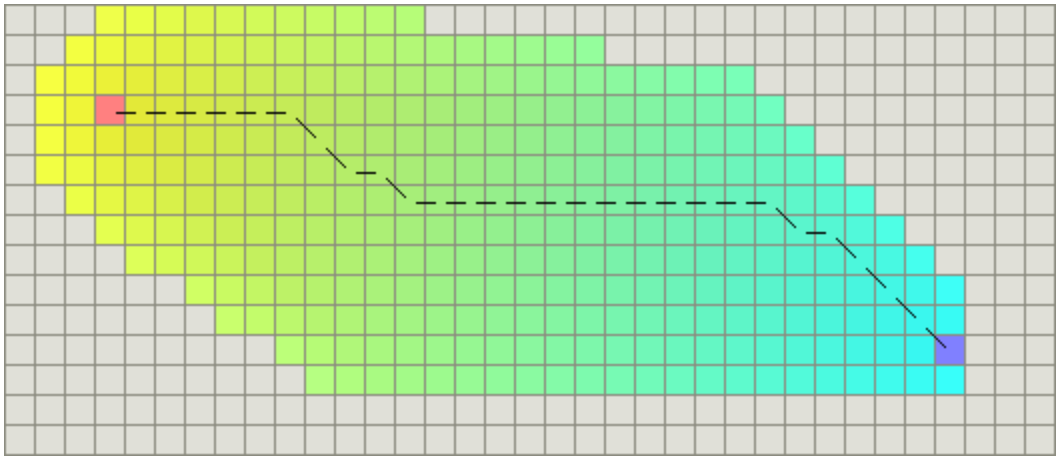
Si se permiten movimientos en diagonal, y s asumien diferente coste para el movimiento diagonal, añadimos los coeficientes que estiman cuántos movimientos horizontales y verticales faltan y cuántos diagonales. Así tenemos una heurística muy informada que sigue siendo admisible.

Heurísticas para mapas “de rejilla”

Con movimientos diagonales

Euclídea

```
function heuristic(node):
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx* dx + dy * dy)
```



D = coste del movimiento

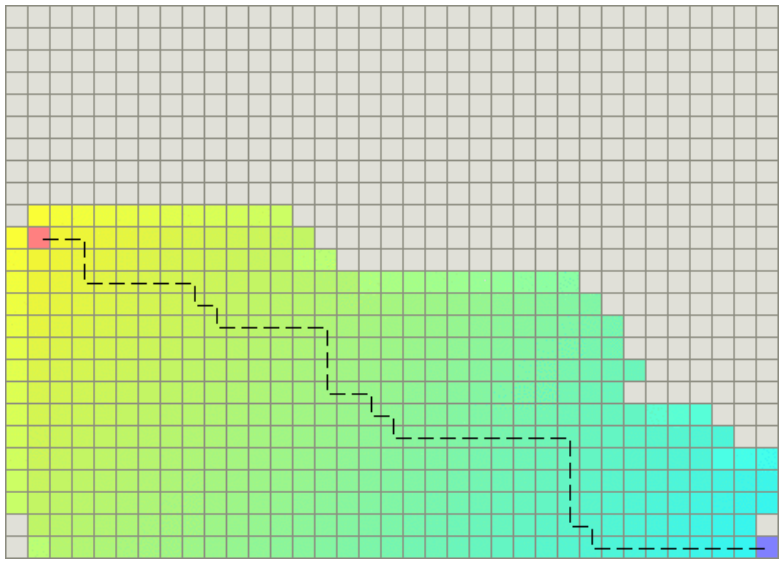
(node.x, node.y)

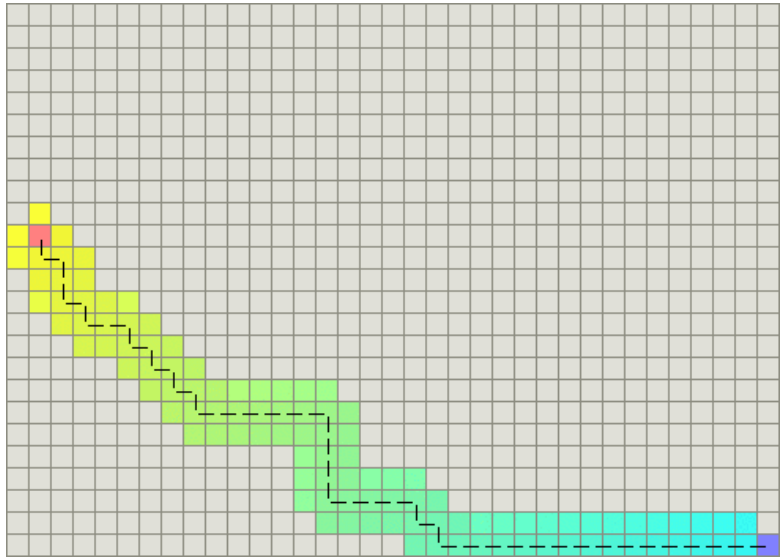
(goal.x, goal.y)

‘Breaking ties’: ¿qué pasa con los empates?

Con movimientos sólo horizontales y verticales

(node.x, node.y)





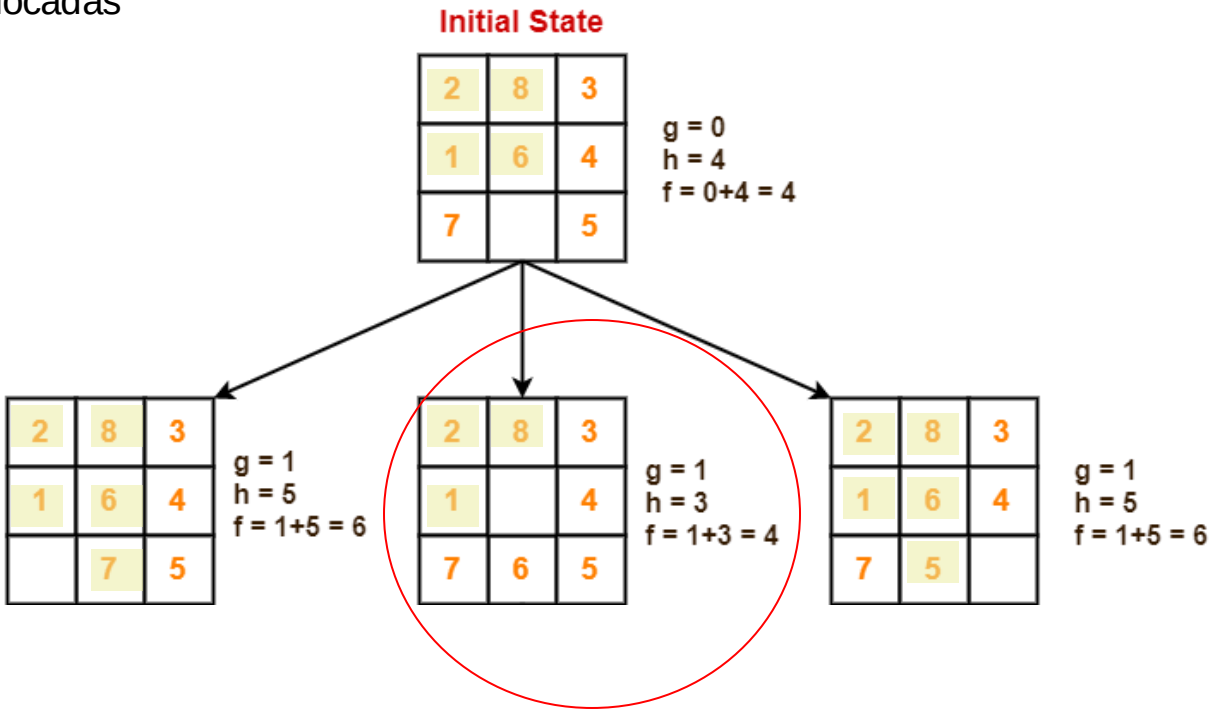
(goal.x, goal.y)

Es discutible el mejor criterio, pero cuando la heurística se sabe que es buena, es habitual optar (en caso de empate en f) por el nodo de mejor heurística (h).

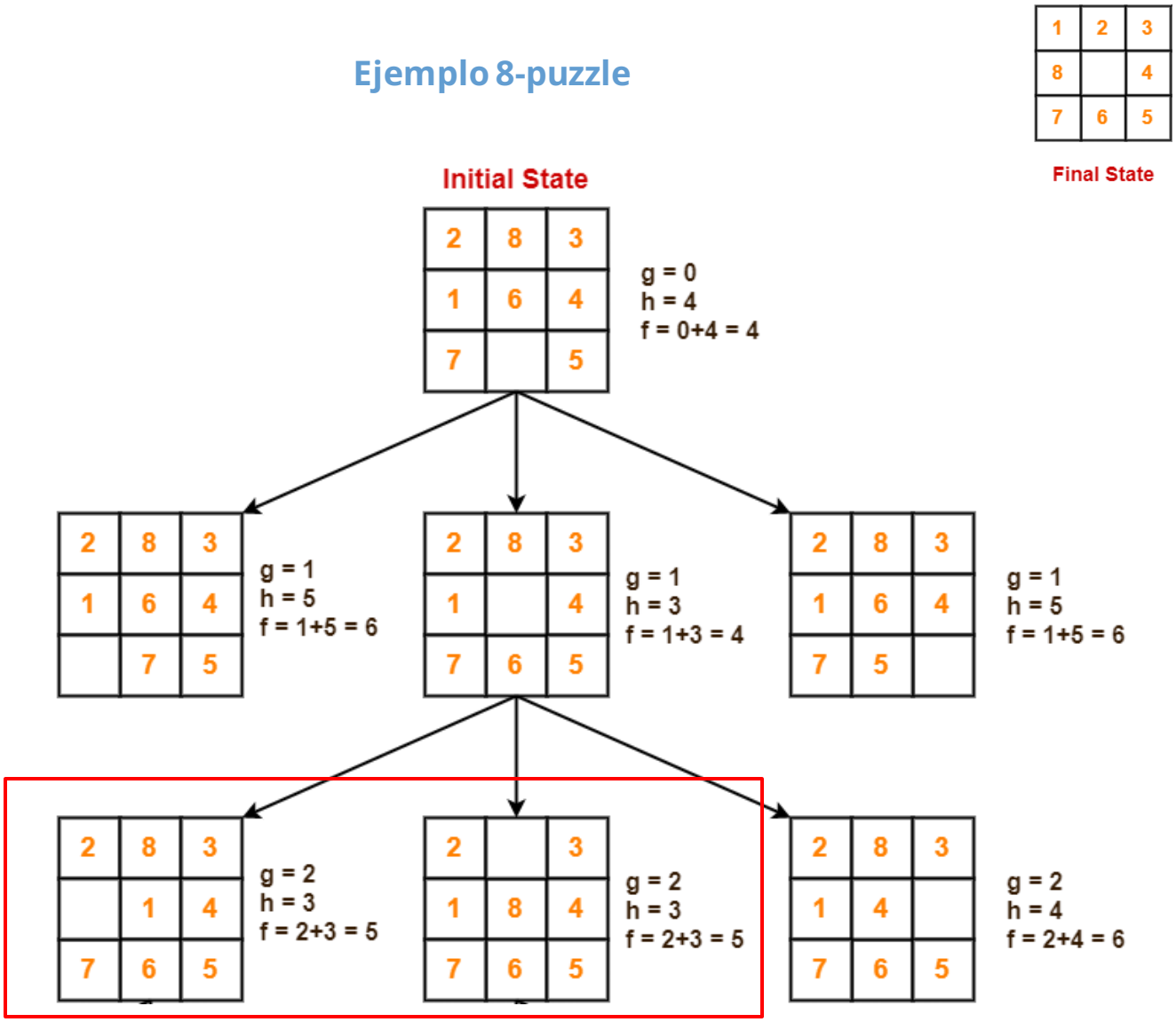
Ejemplo A* adicional

Ejemplo 8-puzzle

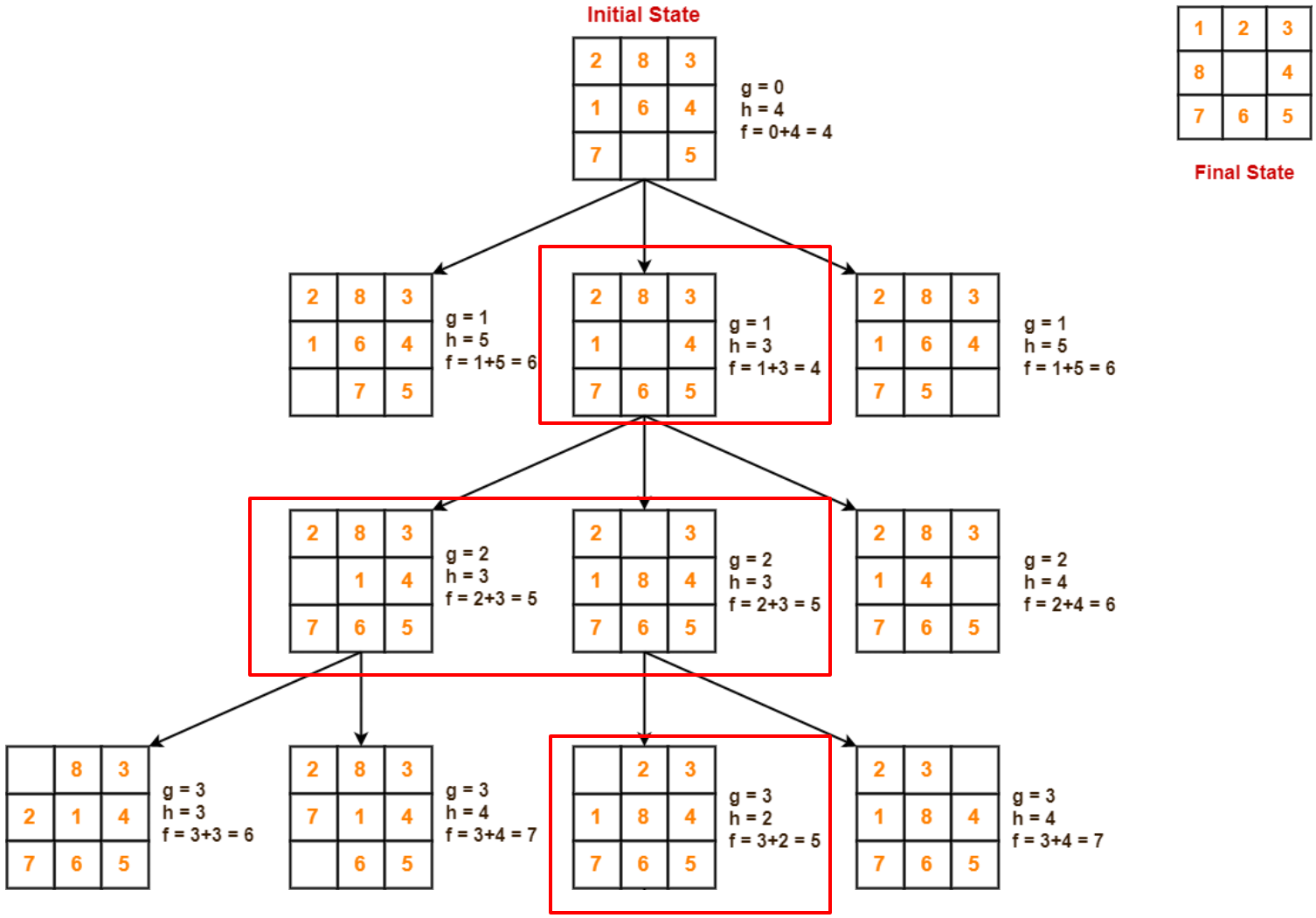
g = 1
h = nº casillas descolocadas



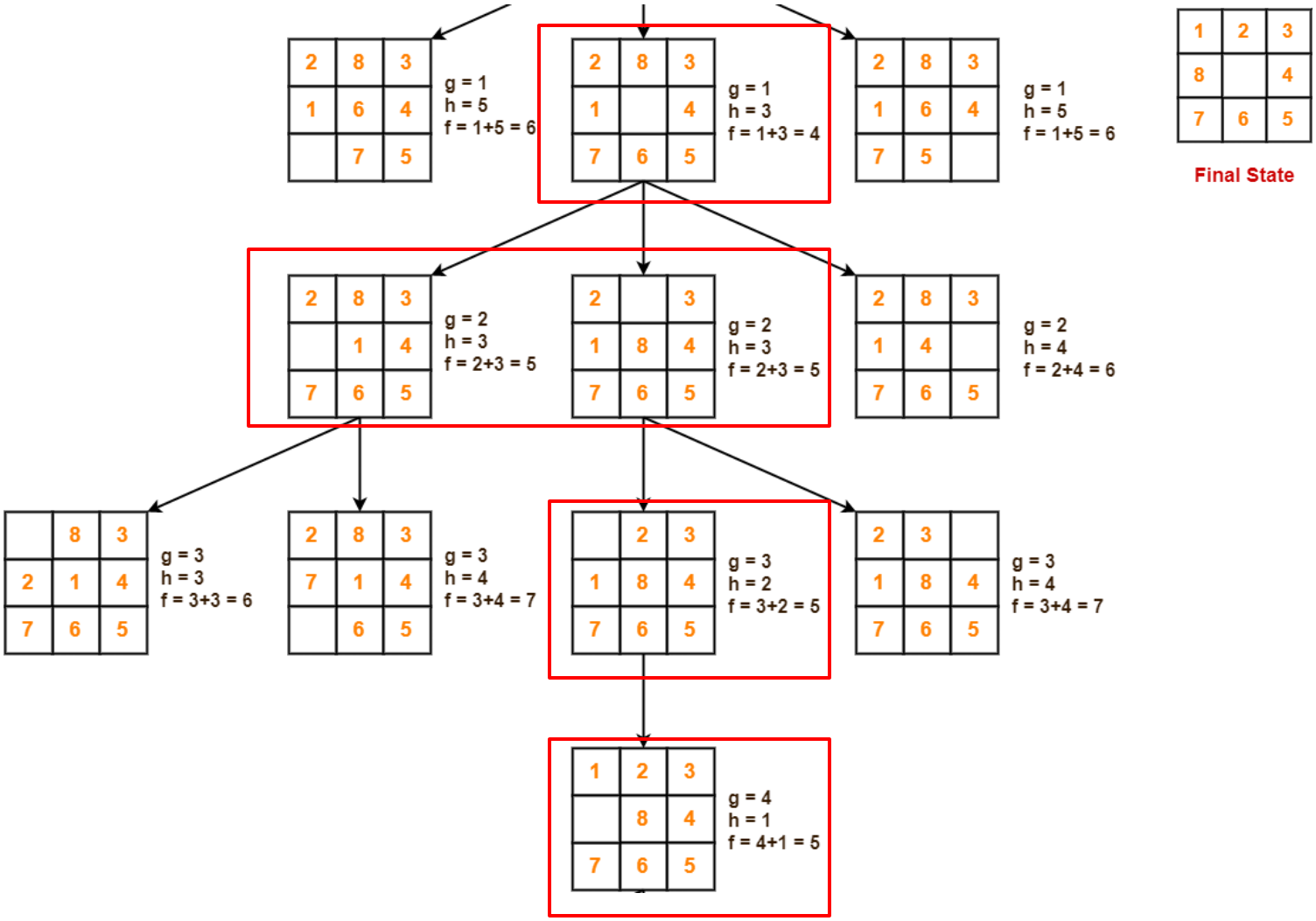
Ejemplo 8-puzzle



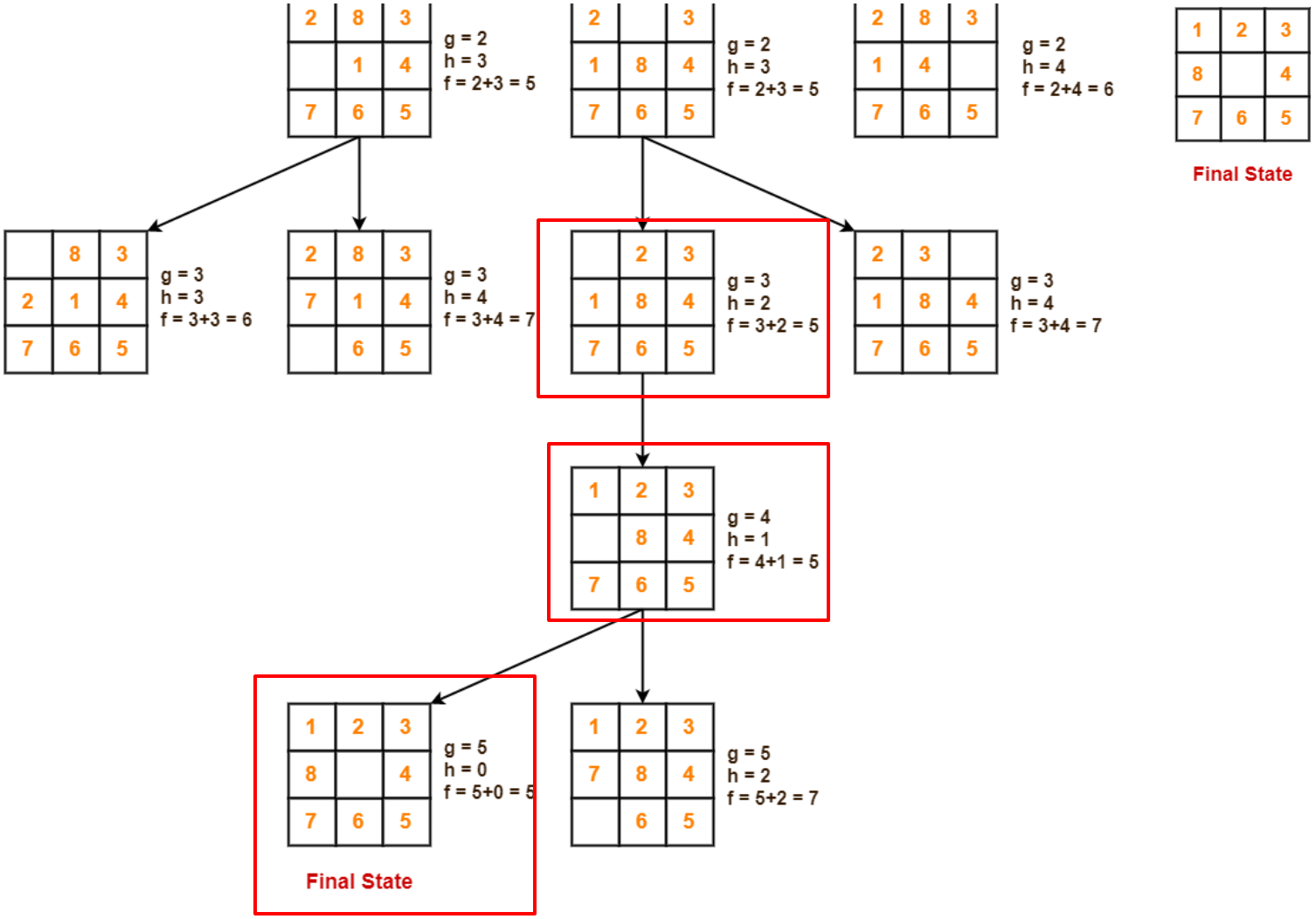
Ejemplo 8-puzzle



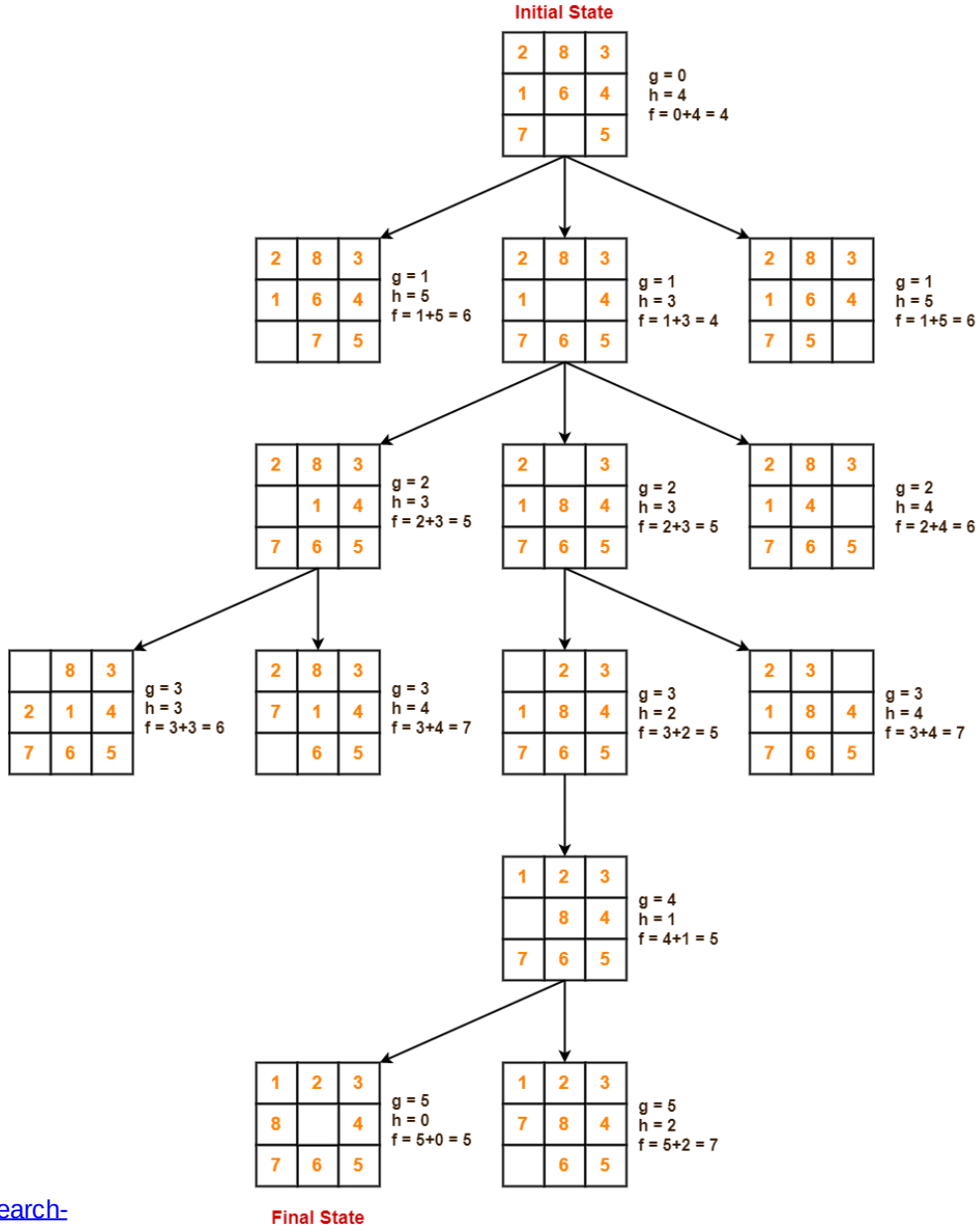
Ejemplo 8-puzzle



Ejemplo 8-puzzle



Ejemplo 8-puzzle



https://www.gatevidyalay.com/tag/a-star-search-algorithm-example/

Ejemplo II 8-puzzle

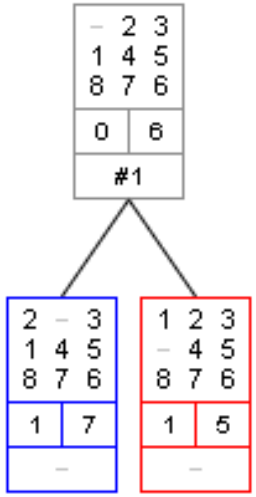
1	2	3
8		4
7	6	5

Final State

g = 1

h = distancia manhattan

Hacer en forma de árbol y comprobar si varía



h= 1+ 1+ 1 + 1 +1 + 1 + 1
(1) (2) (4) (5) (6) (7) (8)

h= 1 + 1 +1 + 1 + 1
(4) (5) (6) (7) (8)

Material para ampliar

- http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#a-stars-use-of-the-heuristic
- https://github.com/aimacode/aima-python
- https://www.youtube.com/watch?v=ySN5Wnu88nE
- https://qiao.github.io/PathFinding.js/visual/
- https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*

www.unir.net