Búsqueda informada

Inteligencia Artificial

Prof: Hugo Franco, PhD

Maestría en Analítica de Datos

Facultad de Ingeniería y Ciencias Básicas

1 de octubre de 2022

Repaso

- **Solución a un problema de búsqueda:** Camino (ruta) desde el estado inicial del problema hasta un estado objetivo
- **Solución óptima:** Aquella solución que tiene el mínimo *costo de ruta* (distancia) dentro de un conjunto de soluciones posibles al problema (presentes en el espacio de estados)
- **Expansión de estados:** Aplicación de cada acción válida (transición, arista) al estado actual, *generando* un nuevo conjunto de estados.
- **Frontera:** Conjunto de vértices del árbol disponibles para ser expandidos hasta un punto dado (iteración) de la exploración del espacio de estados. Son las hojas del árbol hasta donde ha sido construido en la búsqueda.

Búsqueda informada

Búsqueda informada

- Los algoritmos de *búsqueda no informada* son susceptibles a explorar múltiples caminos que **no** conducen a una solución, antes de hallar un estado objetivo en el espacio de estados.
  - Esto aumenta de forma significativa la *complejidad en operaciones* del método (por ende el tiempo de ejecución).

- En problemas de búsqueda, la estrategia debe pasar por la reducción del número de caminos alternativos por evaluar
  - Restringir la expansión de vértices del árbol de búsqueda sobre el grafo que representa el problema.

Búsqueda no informada vs. búsqueda informada

Búsqueda no informada:

- Se parte de la definición del problema (de búsqueda) para usar una estrategia que permita llegar a la solución de la forma más eficiente posible a partir del estado inicial usando solo la exploración mediante la expansión de los nodos.

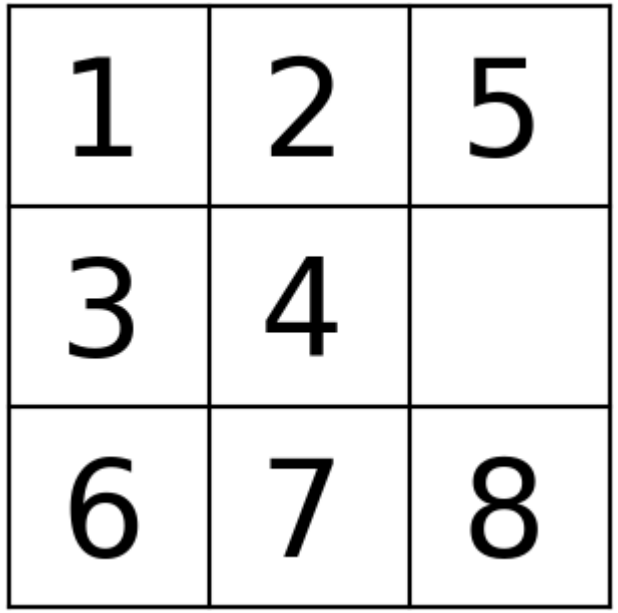
Búsqueda informada:

- Además de la definición del problema, se usa conocimiento previo sobre la naturaleza de la solución.

Esto implica establecer una representación cuantitativa de dicho conocimiento a la hora de escoger las rutas de exploración del espacio de estados.

Ejemplo: análisis de la solución *no informada* - puzzle de ocho piezas

- **Estado inicial:**
  - {(1,2,5),(3,4,X),(6,7,8)} (Ejemplo de la imagen a la derecha).
- **Estados**: todas las posibles configuraciones del puzzle.
- **Acciones:** mover espacio vacío arriba (*Up*), mover abajo (*Down*), mover a la izquierda (*Left*), mover a la derecho (*Right*).
- **Modelo de transiciones**: la casilla vacía no puede salirse de los bordes del tablero.
- **Prueba de éxito:** evaluar si las piezas se encuentran en secuencia ordenada de arriba a abajo y de izquierda a derecho.
- **Función de coste:** todos los movimientos tienen el mismo coste (1). Este se acumula en cada paso.



Ejemplo: estado inicial del puzzle de 8 casillas

Ejemplo: análisis de la solución *no informada* - puzzle de ocho piezas

- En este problema “pequeño”, el coste computacional de una solución no informada (p.ej. BFS o ID-DFS) es relativamente alto, tanto en instrucciones como en memoria.
- Muchas “jugadas” (movimientos, transiciones) pueden identificarse fácilmente como inútiles o incluso contraproducentes en términos del objetivo del sistema/agente.
- Un jugador humano puede establecer, con un poco de observación y experiencia, estrategias que permitan alcanzar más rápidamente la solución.

Heurísticas

Qué es una heurística

- Las **heurísticas** son la *representación* del conocimiento obtenido a través de la experiencia
  - Dicho conocimiento usualmente consiste en **reglas** extraídas de múltiples ejemplos de solución en diferentes escenarios

- Una heurística permite tomar decisiones rápidas sin detenerse en las causas (razones) de cada decisión.
- Cuantas más instancias (ejemplos) de solución sean tenidos en consideración para crear la base de reglas, mejor será la heurística.

Utilidad de las heurísticas

- Con base en heurísticas, es posible obtener soluciones válidas **sin explorar todos los estados posibles del problema**.
  - Las heurísticas permiten traducir el conocimiento experto en mecanismos para hacer más eficiente el proceso de encontrar una solución (reglas, criterios de limitación de la expansión de vértices)

Sin el uso de heurísticas, muchos problemas de búsqueda *no serían tratables* (complejidad en instrucciones y memoria)

- Las heurísticas pueden establecerse a partir de una base de reglas preestablecidas, si bien en trabajos recientes, pueden aprenderse mediante técnicas modernas de aprendizaje de máquina (Deep Learning)

Formalización de las heurísticas

- La expansión de un vértice no se hace arbitrariamente (como en la búsqueda no informada) sino que se establece una función de evaluación $f(v)$
  - v usualmente es el vértice bajo evaluación en cada paso del proceso de exploración en el algoritmo de búsqueda
  - La selección de $f(v)$ establece la **estrategia de búsqueda**.
- Esta función provee una estimación del costo
  - No se reduce a costos preestablecidos en la representación durante la exploración
- La mayoría de algoritmos basados en “primero - *” (selección de una ruta óptima con conocimiento parcial) se basan en la definición de la función heurística, denotada $h(v)$:
  - Costo estimado de la ruta más “económica” desde el estado actual (en v) hasta un estado solución

Algunos algoritmos de búsqueda informada

Algunos algoritmos de solución de problemas de búsqueda informada (heurísticos) en Inteligencia Artificial son:

- **Búsqueda “primero-mejor” (“voraz”)**
- **Algoritmo de búsqueda A***
- Búsqueda heurística “acotada en memoria”

Algoritmos de búsqueda informada

Búsqueda “primero-el-mejor”

Búsqueda “primero-el-mejor”

- El algoritmo “primero-el-mejor” se dice “*voraz*” porque usa la misma estrategia de escoger siempre para avanzar (de forma local) el vértice de la expansión con el menor coste.

- La diferencia es que el "coste" (o, en general, la función de evaluación) se estima en función de una heurística $h\,(v)$, en lugar del valor de coste estricto de las transiciones durante la búsqueda, $g\,(v)$.

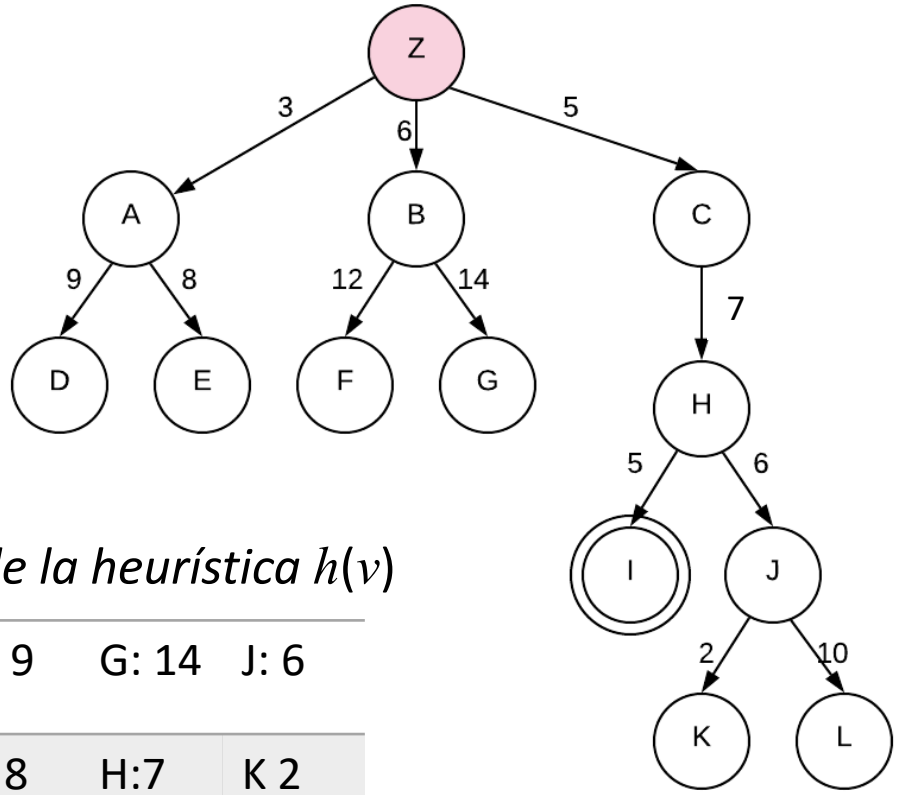
Algoritmo “primero-mejor”

1. Ingresar el nodo inicial v_0 y su valor heurístico $h\ (v_0)$ en la lista de evaluación, denominada S (la frontera, implementada como una **cola de prioridad, los elementos se introducen ordenados**)
2. Tomar un nodo v desde la parte superior de la frontera (S).
   a) Si v es el nodo objetivo, reconstruir la solución (trayectoria).
   b) De lo contrario, si la lista está vacía, retornar fallo.
3. Agregar v a la lista de visitados y expandir v, generando el conjunto de nodos de expansión v’.
4. Para cada estado expandido v’
   a) Si v' no está en la frontera S ni en la lista de visitados, introducir v‘ en S de forma ordenada según el valor de h, junto con la arista (v, v’)*.
   b) De lo contrario**, si** v’ **está en la frontera pero no en la lista de visitados, si** $h(v\text{'})$ **es menor que el valor anterior,** actualizar v' con la nueva arista y el nuevo valor de la heurística.
5. Regresar al Paso 2.

* Se usa para especificar el padre de cada nodo

Algoritmo “primero-mejor” - Ejemplo

Problema: encontrar la ruta de S hasta I



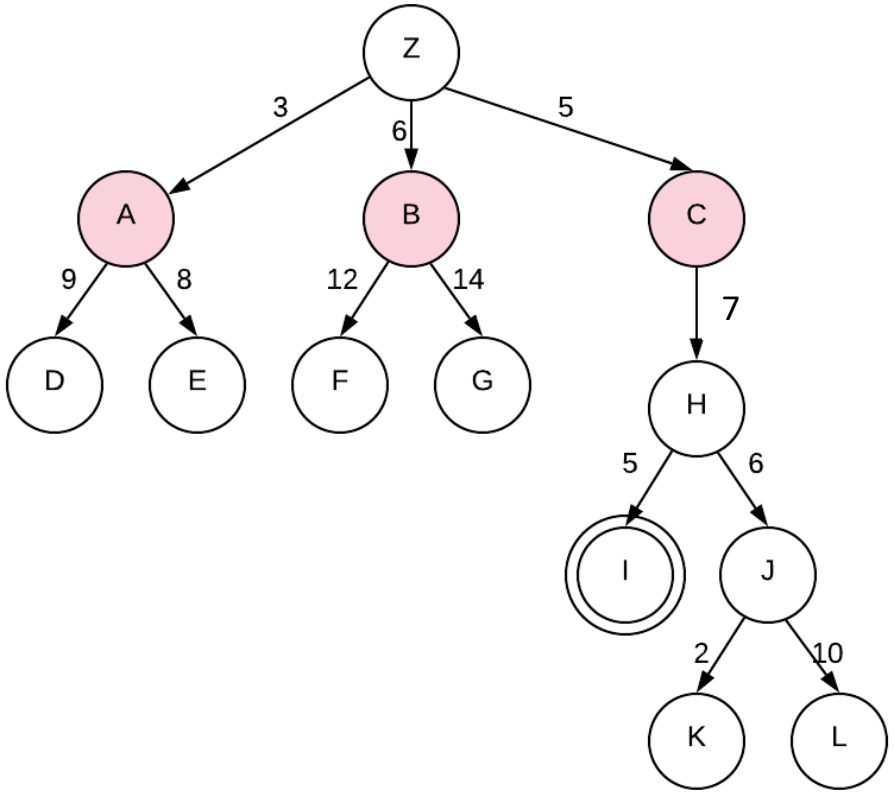
Valores de la heurística $h(v)$

A: 3	D: 9	G: 14	J: 6
B: 6	E: 8	H:7	K 2
C: 5	F: 12	I: 5	L:10

- En las etiquetas de las aristas se especifica $h\ (v)$ cuando el estado inicial es Z
- Se comienza desde el estado inicial (v = Z) y se busca el objetivo "I“, usando los costos dados y la **búsqueda primero-mejor**.
- Primera iteración, se agrega Z a la lista de **evaluación**.
- Como Z no es el estado solución, se continua

Algoritmo “primero-mejor” - Ejemplo

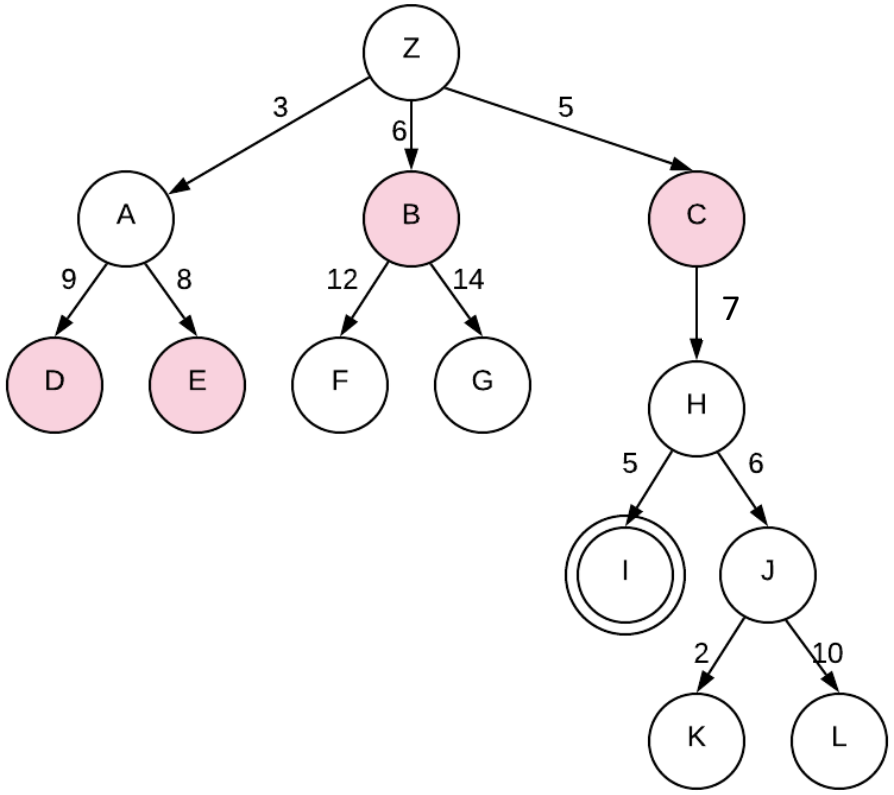
Problema: encontrar la ruta de S hasta I



- Se retira v (esto es, Z) de la lista de evaluación y se agrega a la lista de visitados, {Z}
- Se añaden los vecinos de Z al conjunto S (esto es, {A, C, B})
  - C se pone adelante de B en la lista de prioridad por tener un costo menor)
- Como S no es vacía y ningún estado en ella es solución, se continúa

Algoritmo “primero-mejor” - Ejemplo

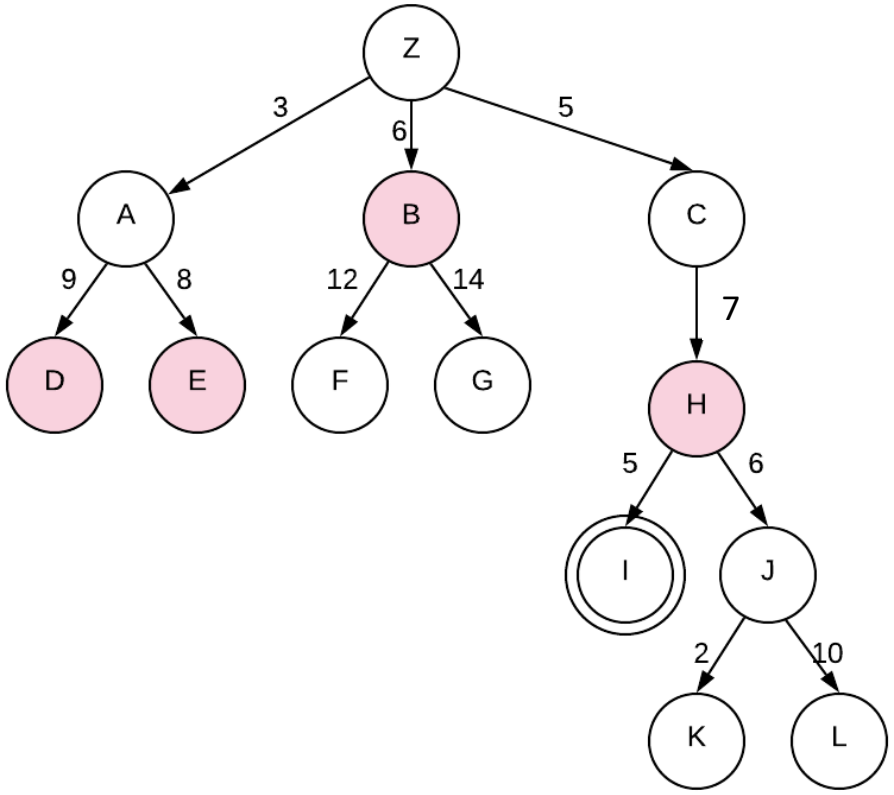
Problema: encontrar la ruta de S hasta I



- Se retira *v* (esto es, A) de la lista de evaluación, se agrega a la lista de visitados {Z, A}.
- Se agregan los vecinos de A a la lista de evaluación S, que queda como {C, B, E, D}
  - E va primero por tener menor valor de $h\ (v)$
- Como S no es vacía y ningún estado en ella es solución, se continúa

Algoritmo “primero-mejor” - Ejemplo

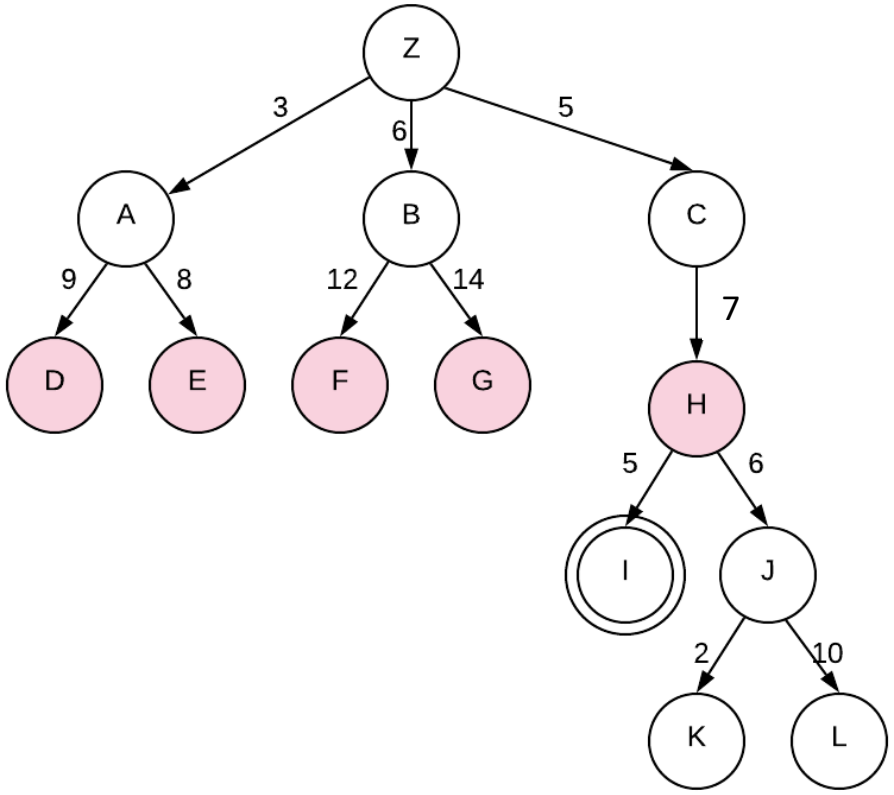
Problema: encontrar la ruta de S hasta I



- El siguiente en la lista de evaluación es C, luego se elimina de allí y se agrega a la lista de visitados {Z, A, C} (se requiere para evitar repetidos y, por ende, circuitos en grafos)
- Se agregan los vecinos de C a la lista de evaluación
- La lista S ahora contiene {B, H, E, D}
- Como S no es vacía y ningún estado en ella es solución, se continúa

Algoritmo “primero-mejor” - Ejemplo

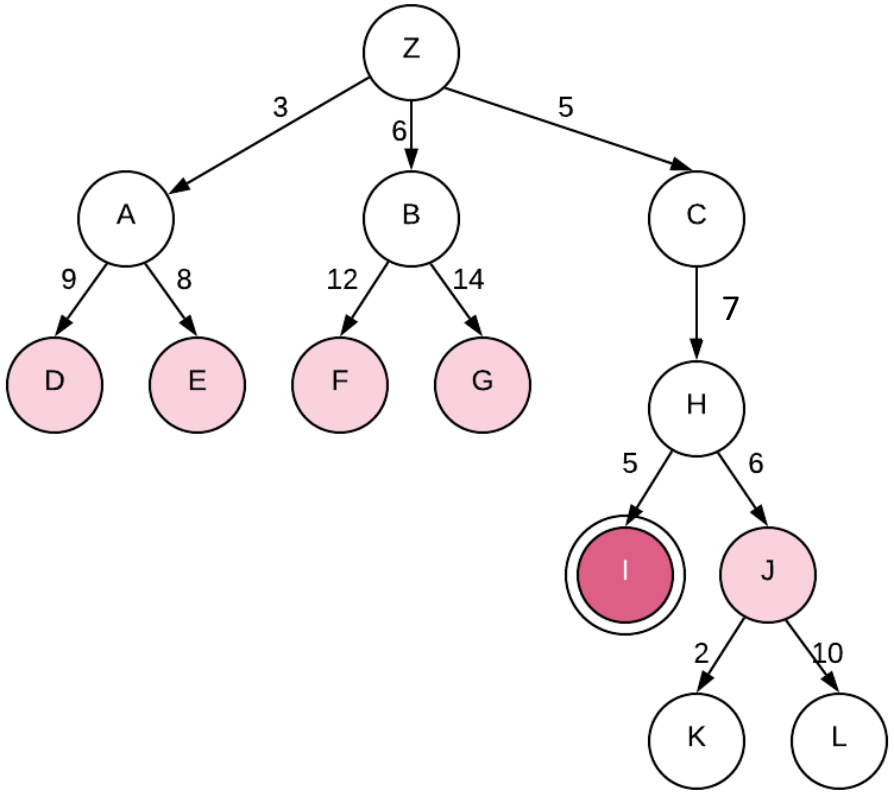
Problema: encontrar la ruta de S hasta I



- El siguiente en la lista de evaluación S es B, luego se elimina de ella y se agrega a la lista de visitados {Z, A, C, B}
- Se agregan los vecinos de B a la lista de evaluación, que ahora queda como {H, E, D, F, G}
- Como S no es vacía y ningún estado en ella es solución, se continúa

Algoritmo “primero-mejor” - Ejemplo

Problema: encontrar la ruta de S hasta I



- Se retira H de la lista de evaluación y se agrega a la lista de visitados {Z, A, C, B, H}
- Se agregan los vecinos de H, luego la lista de evaluación queda como {I, J, E, D, F, G}
- Se continúa a la siguiente iteración, como “I” es un estado objetivo, al evaluarse termina el algoritmo y se retorna “I” como solución
  - Nota: el paso 4 (b) agrega a cada nodo la lista de recorrido que viene de su nodo padre, permitiendo recuperar el recorrido

Algoritmos de búsqueda informada

Algoritmo A*

Formulación del Algoritmo A*

- No es posible garantizar la obtención de la solución óptima utilizando la búsqueda "primero mejor".
  - Esto se debe a que se ignora el costo real desde el nodo inicial hasta el nodo actual.
- El algoritmo A* fue diseñado para abordar este problema. La letra A se suele interpretar como “admisible”.
- En el algoritmo A*, el costo de un nodo se evalúa utilizando tanto el costo estimado como el costo real según la expresión:

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

  donde g corresponde al coste real en el grafo (coste de la ruta desde el nodo inicial hasta el nodo v)

- Se ha demostrado que el algoritmo A* puede obtener la mejor solución siempre que el costo estimado $h(v)$ sea siempre menor que el mejor valor posible $h^*(v)$.

Algoritmo A*

1. Poner el nodo inicial v_0 y su costo $f(v_0) = h(v_0)$ en la frontera S (también una *cola de prioridad*). Inicializar la "lista de visitados" *V* a vacío.
2. Tomar un nodo v desde la cabeza de S.
   a) Si v es el nodo objetivo, **retornar** el estado objetivo (éxito).
   b) De lo contrario, si la frontera S está vacía, **retornar** error.
3. Expandir v para agregar a S los hijos de v. Poner v en la lista de visitados.
4. Para todo v' en S, calcular su costo
   a) Si v' está en la lista de visitados (*V*) pero el nuevo costo $f(v')$ es menor que el anterior, mover v' de la lista de visitados a la **frontera** S y actualizar la arista (v, v')* y su costo.
   b) De lo contrario, si v' está en la **frontera** S, pero el nuevo costo $f(v')$ es menor que el anterior, actualizarla con la arista (v, v') y su costo.
   c) De lo contrario (si v' no está en la frontera *S* ni en la lista de visitados *V*), agregar v', la arista (v, v') y el costo $f(v')$ a la lista de evaluación.
5. Regresar al paso 2.

* Se usa para especificar el padre de cada nodo

Ejemplo: puzzle de 8 piezas

- Estado inicial

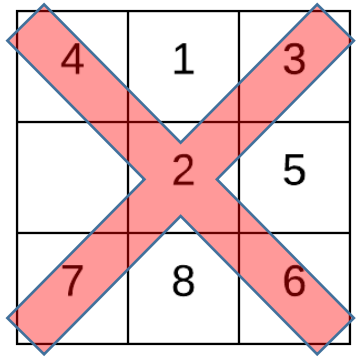
4	1	3
	2	5
7	8	6

- Estado final

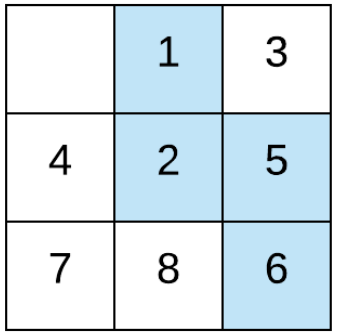
1	2	3
4	5	6
7	8	

- El algoritmo comienza agregando el estado inicial en la lista de evaluación, S={Z}
- La función de costo por transición, como fue mencionado, puede asumirse como $g(v) = 1$ para cada nueva transición (se acumulan a lo largo de la ruta)
- La función heurística $h\ (v)$ puede definirse de multiples formas, p. ej. el conteo de casillas mal ubicadas de acuerdo con el estado de éxito buscado.

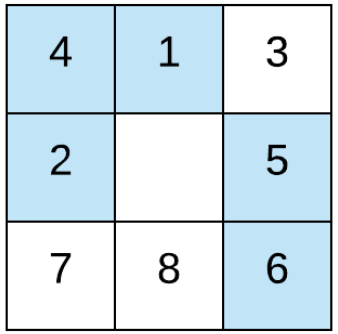
Algoritmo A* - Ejemplo



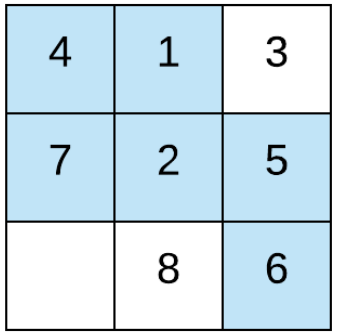
Z



A {Z}
h(A)=4
f(A)=1+4=5



B {Z}
h(B)=5
f(B)=1+5=6

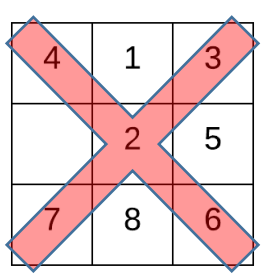


C {Z}
h(C)=6
f(C)=1+6=7

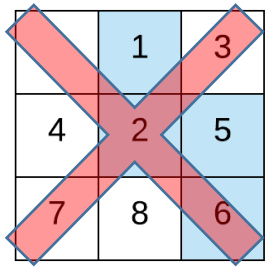
- Retirar el primer estado Z de la lista de evaluación y ponerlo en la lista de visitados, *V*={Z}
- Expandir los nodos vecinos (esto es, aplicar las transiciones posibles) para **generar** la frontera S en la representación seleccionada. Así pues, S = {A, B, C}
- *f* se calcula como es usual:

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

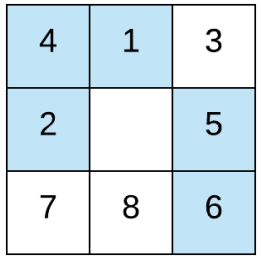
Algoritmo A* - Ejemplo



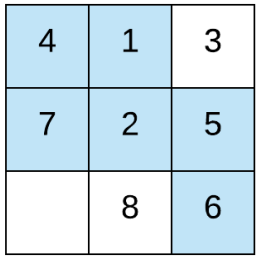
Z



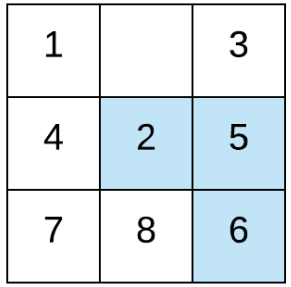
A {Z}
h(A)=4
f(A)=5



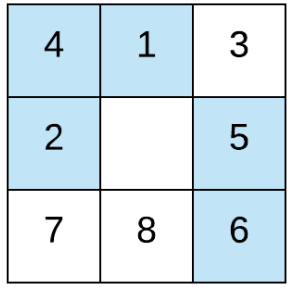
B {Z}
h(B)=5
f(B)=6



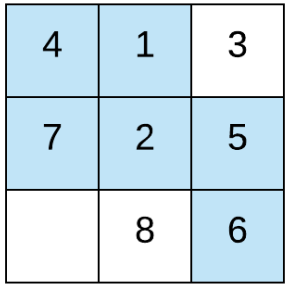
C {Z}
h(C)=6
f(C)=7



D {Z A}
h(D)=3
f(D)=2+3=5



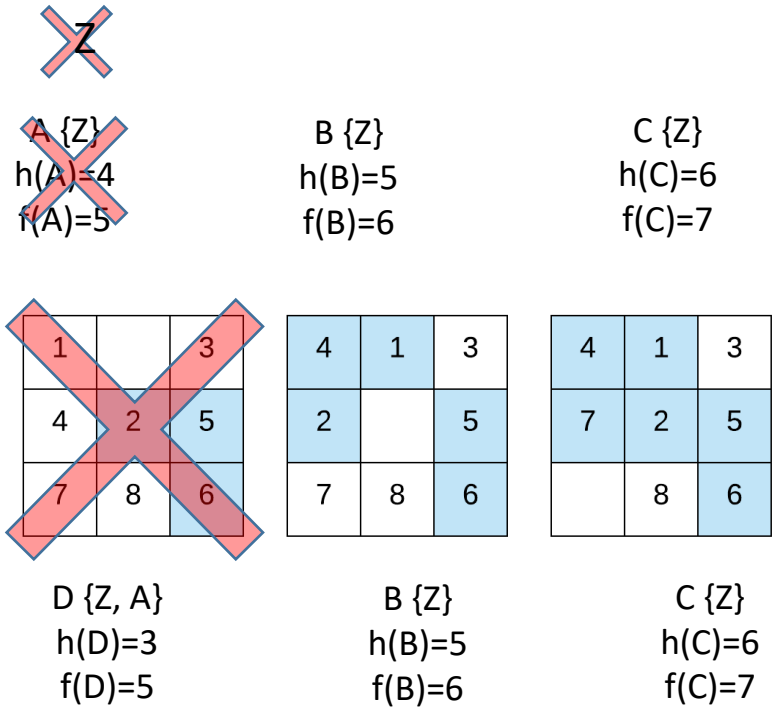
B {Z}
h(B)=5
f(B)=6



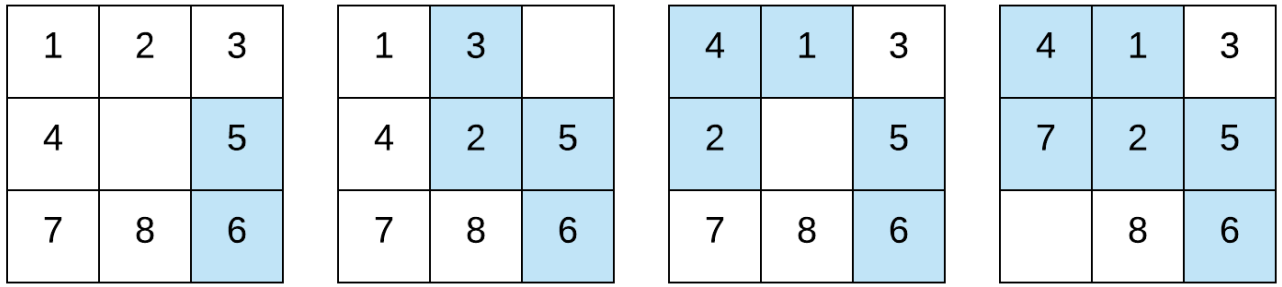
C {Z}
h(C)=6
f(C)=7

- El primer lugar en la lista de evaluación lo tiene A, $f(A) = 5$. Se retira y se pone en la lista de visitados, que queda *V*= {Z,A}
- Expandir los estados vecinos para generar la frontera S (teniendo en cuenta los costos de los visitados). Así, se genera el estado D
- El estado D tiene función de costo 5, pues g (D) =2. La frontera queda como S ={D, B, C}

Algoritmo A* - Ejemplo



- D, primero en la frontera S ($f = 5$) se retira y se pone en la lista de visitados, V= {Z, A, D}
- Se expanden los vecinos de D para generar la lista de evaluación (teniendo en cuenta los costos de los visitados). Así, se generan los estados E y F, cuyo costo para $g(v)$ es 3.
- El estado E tiene función de costo 5, mientras que F tiene costo de 7. Así, S = {E, B, C, F}



Algoritmo A* - Ejemplo

Z

A {Z}
h(A)=4
f(A)=5

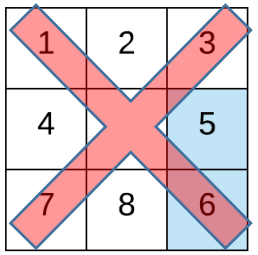
B {Z}
h(B)=5
f(B)=6

C {Z}
h(C)=6
f(C)=7

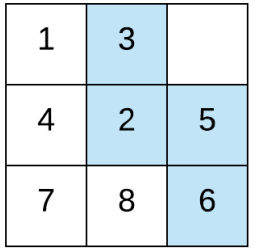
D {Z, A}
h(D)=3
f(D)=5

B {Z}
h(B)=5
f(B)=6

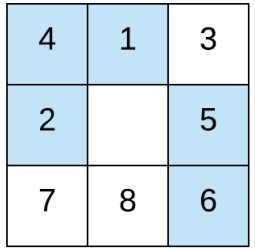
C {Z}
h(C)=6
f(C)=7



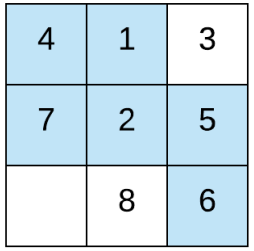
E {Z, A,D}
h(E)=2
f(E)=5



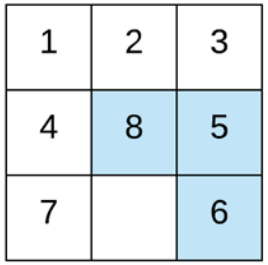
F {Z, A}
h(F)=4
f(F)=7



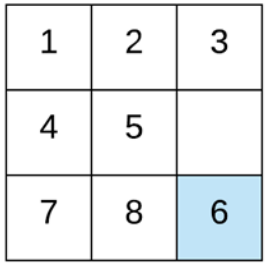
B {Z}
h(B)=5
f(B)=6



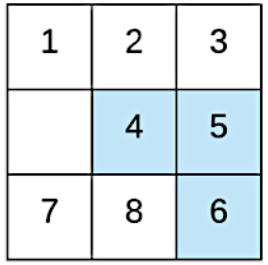
C {Z}
h(C)=6
f(C)=7



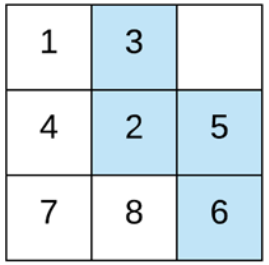
G {Z, A, D. E}
h(G)=3
f(G)=4+3=7



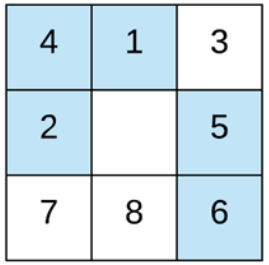
H {Z, A, D. E}
h(H)=1
f(H)=4+1=5



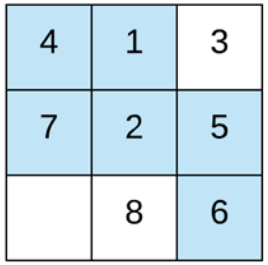
I {Z, A, D. E}
h(I)=3
f(I)=4+3=7



F {Z, A}
h(F)=4
f(F)=7



B {Z}
h(B)=5
f(B)=6



C {Z}
h(C)=6
f(C)=7

- E, primero en la lista de evaluación ($f = 5$) se retira y se pone en la lista de visitados, V={Z, A, D, E}
- Se expanden los vecinos de E, la nueva lista de evaluación. Así, se generan los estados G, H e I, con $g\,(v) = 4$.
- La frontera queda como S = {H, B, C, F, G, I}

Algoritmo A* - Ejemplo

Z

A {Z}
h(A)=4
f(A)=5

B {Z}
h(B)=5
f(B)=6

C {Z}
h(C)=6
f(C)=7

D {Z, A}
h(D)=3
f(D)=2+3=5

B {Z}
h(B)=5
f(B)=6

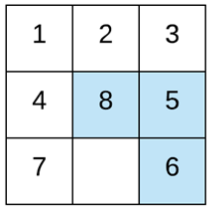
C {Z}
h(C)=6
f(C)=7

E {Z, A,D}
h(E)=2
f(E)=5

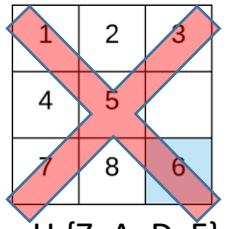
F {Z, A}
h(F)=4
f(F)=7

B {Z}
h(B)=5
f(B)=6

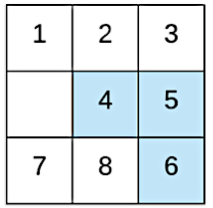
C {Z}
h(C)=6
f(C)=7



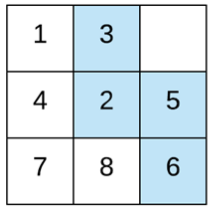
G {Z, A, D. E}
h(G)=3
f(G)=4+3=7



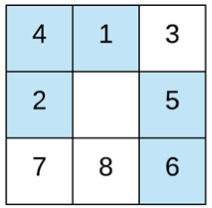
H {Z, A, D, E}
h(H)=1
f(H)=4+1=5



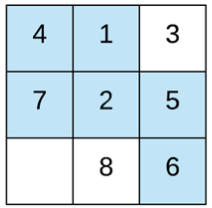
I {Z, A, D. E}
h(I)=3
f(I)=4+3=7



F {Z, A}
h(F)=4
f(F)=7

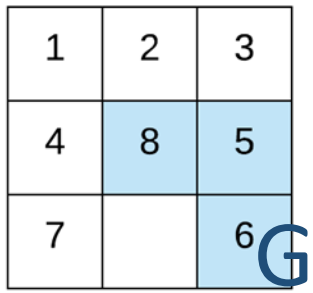


B {Z}
h(B)=5
f(B)=6

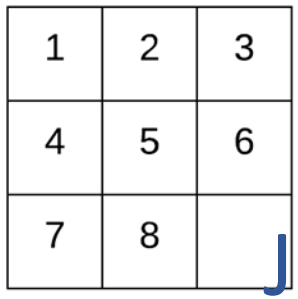


C {Z}
h(C)=6
f(C)=7

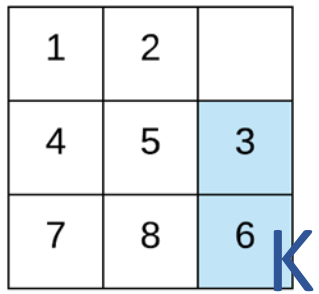
- H, primero en la frontera S ($f = 5$) se retira de S y se pone en la lista de visitados, V={Z, A, D, E, H}
- Se expanden los vecinos de H:
  - J, con: $f(\mathrm{J}) = h(\mathrm{J}) + g(\mathrm{J}) = 0 + 5 = 6$
  - K, con: $f(\mathrm{K}) = h(\mathrm{K}) + g(\mathrm{K}) = 2 + 5 = 8$
- La frontera ahora es S={J, B, C, F, G, I, K}



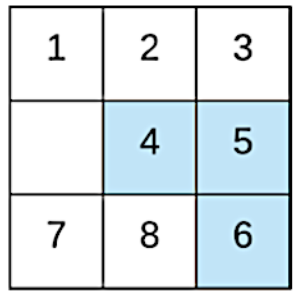
G



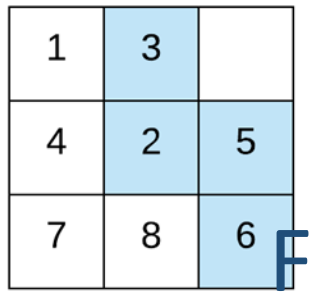
J



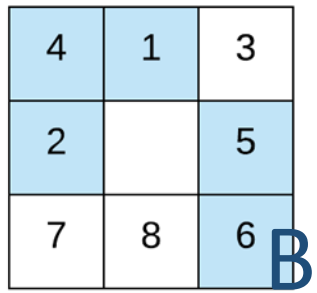
K



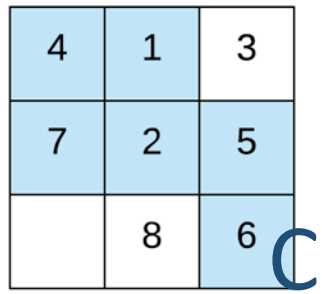
I



F



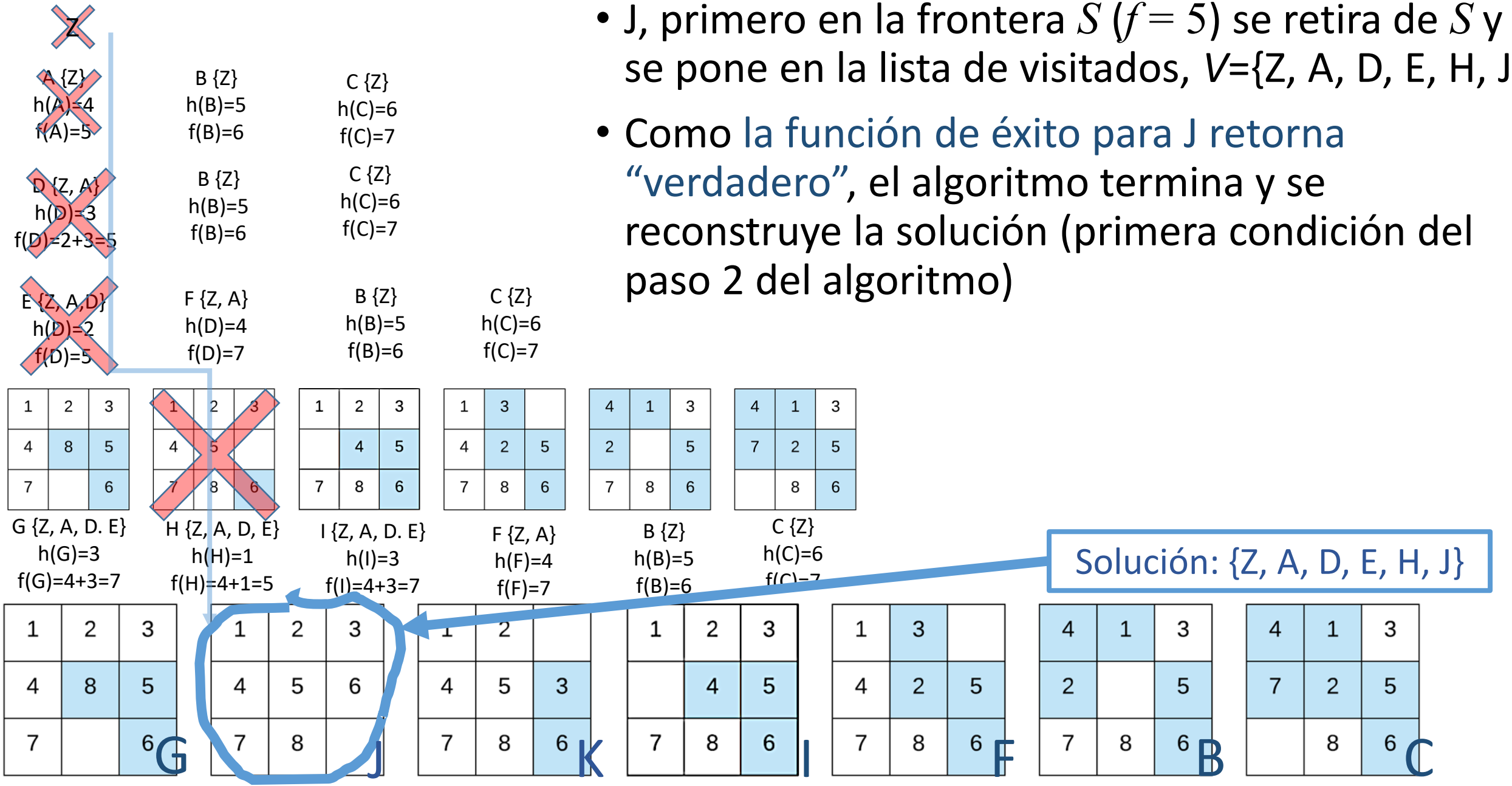
B



C

Algoritmo A* - Ejemplo

- J, primero en la frontera S ($f = 5$) se retira de S y se pone en la lista de visitados, V={Z, A, D, E, H, J}
- Como la función de éxito para J retorna "verdadero", el algoritmo termina y se reconstruye la solución (primera condición del paso 2 del algoritmo)



Criterios de validez y optimalidad

Admisibilidad de una heurística y A*

- Una heurística es **admisible** si se cumple que $h(v)$ nunca sobreestima el costo real de llegar al estado objetivo (éxito) desde el estado v.
  - El valor de la heurística es menor o igual que el costo real del camino al estado objetivo

- Dado que $g(v)$ es el costo real de llegar desde el vértice raíz (estado inicial), entonces:

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

si la heurística es admisible, nunca estará por encima del costo real de alcanzar el resultado desde v.

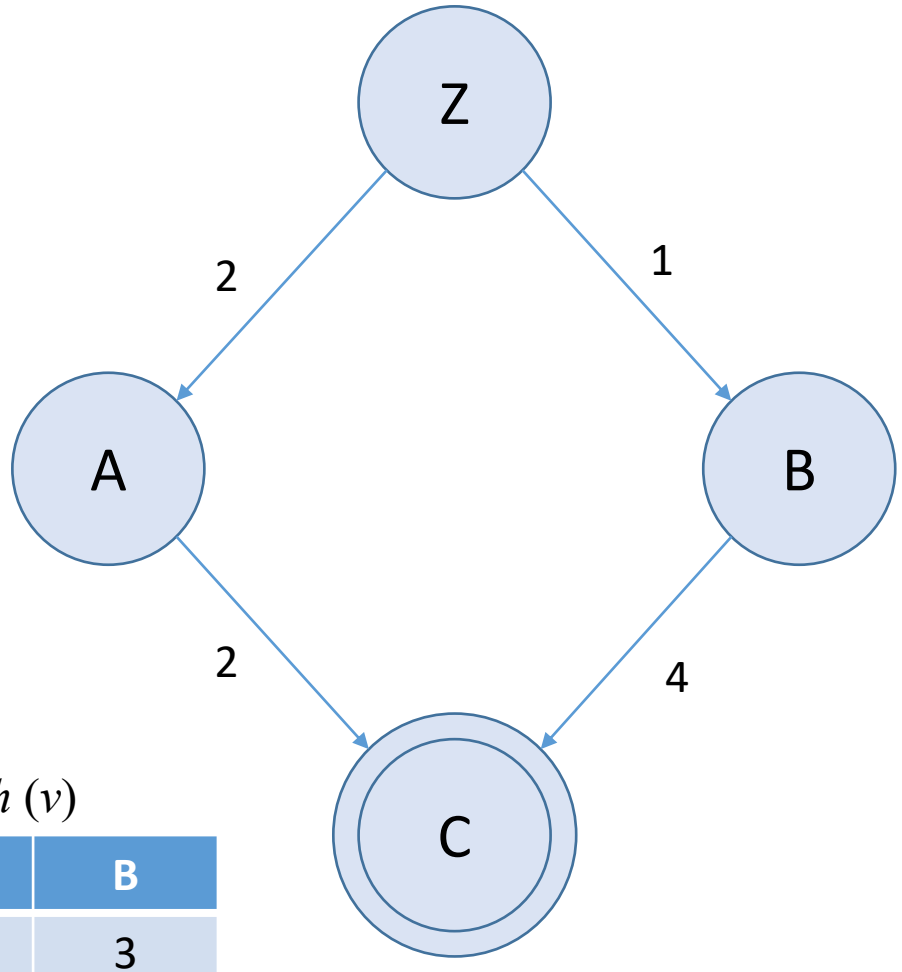
Optimalidad de A*

- Si A* es empleado para resolver un problema de búsqueda usando una heurística admisible $h(v)$ sobre una representación en árbol, entonces la solución obtenida por el algoritmo es óptima.
- Se dice que una heurística admisible es optimista porque siempre asigna un valor menor al costo real de llegar al estado objetivo.
- Dadas dos heurísticas tales que

$$h_2(v) > h_1(v)$$

para todo v, siempre es más eficiente usar $h_2(v)$
  - Se dice que h_2 es más realista (más informada).

Admisibilidad de una heurística - Ejemplo



$h(v)$

A	B
1	3

- La heurística $h(v)$ es admisible puesto que

$$f(A) = 2 + 1 = 3$$
$$f(B) = 1 + 3 = 4$$

- En ambos casos, se cumple que el valor de la heurística $h(v)$ está por debajo del costo real de llegar al estado objetivo desde v, C (A=2, B=4)
- El algoritmo A* preferirá la ruta A que, en este ejemplo, es óptima
  - Costo del camino {Z, A ,C} es 4
  - Costo del camino {Z, B, C} es 5

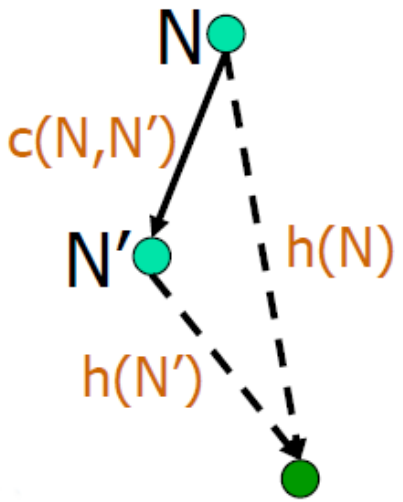
Monotonicidad de una heurística y A*

- Una heurística es **monotónica** (o "consistente") si se cumple que para todo par de estados v y v' tal que v' es sucesor de v, se tiene que $h(v)$ es menor que la suma del costo de la transición de v a v' -peso de la arista (v, v')- más la heurística en el nodo v':

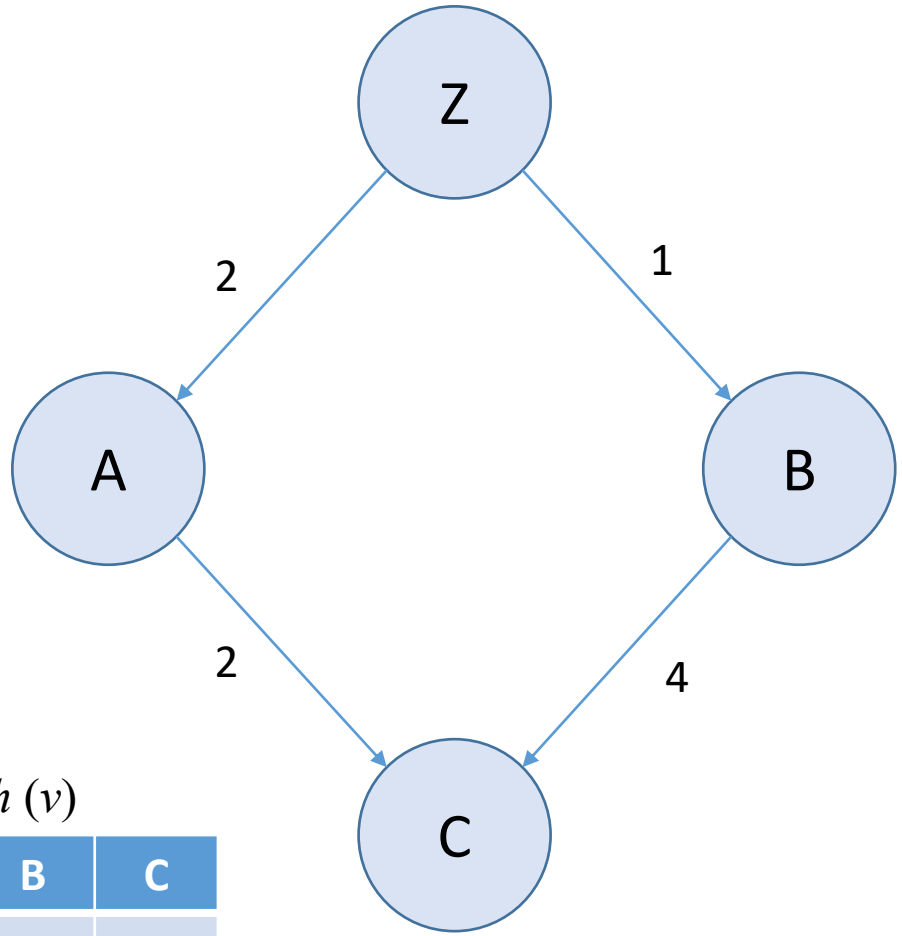
$$\forall v, v' :\in G \quad h(v) \leq c(v, v') + h(v')$$

- Esto equivale a una desigualdad triangular, que tiende a la elección de caminos de menor coste con una aproximación optimista

- Una heurística monotónica siempre es admisible (la monotonicidad es una condición más fuerte que la admisibilidad)
- En una representación en grafo, A* siempre retornará la solución óptima si $h(v)$ es monotónica



Monotonicidad de una heurística - Ejemplo



$h(v)$

A	B	C
1	3	2

- La heurística $h(v)$ es monotónica. Por ejemplo

$$h(\text{A}) = 1$$

$$c(\text{A, C}) = 2 \text{ y } h(\text{C}) = 2$$

$$1 = h(A) \leq c(\text{A, C}) + h(\text{C}) = 2 + 2 = 4$$

$$h(\text{B}) = 3$$

$$c(\text{B, C}) = 4 \text{ y } h(\text{C}) = 2$$

$$3 = h(\text{B}) \leq c(\text{B, C}) + h(\text{C}) = 4 + 2 = 6$$

¿Cuándo usar estrategias de solución a problemas de Inteligencia Artificial mediante búsqueda?

- El espacio de estados es relativamente *pequeño* y además
  - No existen otras aproximaciones viables a mano
  - No compensa desarrollar una técnica más eficiente

- El espacio de estados es *grande* y además
  - No existen otras aproximaciones viables a mano
  - Se pueden establecer heurísticas apropiadas en términos de información disponible, experiencia y simplicidad

Los métodos de solución de problemas siempre tendrán complejidades computacionales altas, $O(b^n)$, con b el factor de ramificación (transiciones posibles) y n la profundidad máxima