Fundamentos de Inteligencia Artificial

Centro Asociado de Melilla
Tutor: Aziz Mulud Tieb

Bloque 3:

Técnicas Basadas en Búsquedas Heurísticas

3.1 Concepto de búsqueda heurística

3.2 Primero el mejor

3.3 Algoritmo A*

3.4 Algoritmos voraces

3.5. Algoritmos de ramificación y poda

3.6. Algoritmos de escalada o máximo gradiente

3.7. Búsqueda tabú

3.1- Concepto de búsqueda heurística

- Las técnicas de búsqueda heurística usan el conocimiento del dominio para adaptar el solucionador y, de esta manera, éste sea más potente y consiga llegar a la solución con mayor rapidez.
- Reducir anchura del árbol expandido: **eliminar algunas ramas** porque probablemente no reflejan la solución óptima
- Estas técnicas utilizan el conocimiento para avanzar buscando la solución al problema.
- **Definiciones**:
  - **Costo del camino**: coste necesario para ir del nodo raíz al nodo meta por dicho camino.
  - **Costo para hallar la solución**: coste necesario para encontrar el camino anteriormente definido.
- **Potencia heurística**: capacidad de un método de exploración para obtener la solución con un coste lo más bajo posible.
- La principal diferencia de este tipo de búsqueda es que ahora a cada nodo se le va a poder asociar un valor que dará idea de lo cerca que se encuentra de un nodo meta.

Función de evaluación heurística

- **Definición**: es una aplicación del espacio de estados con el espacio de los números reales:

$$F(estado) = n$$

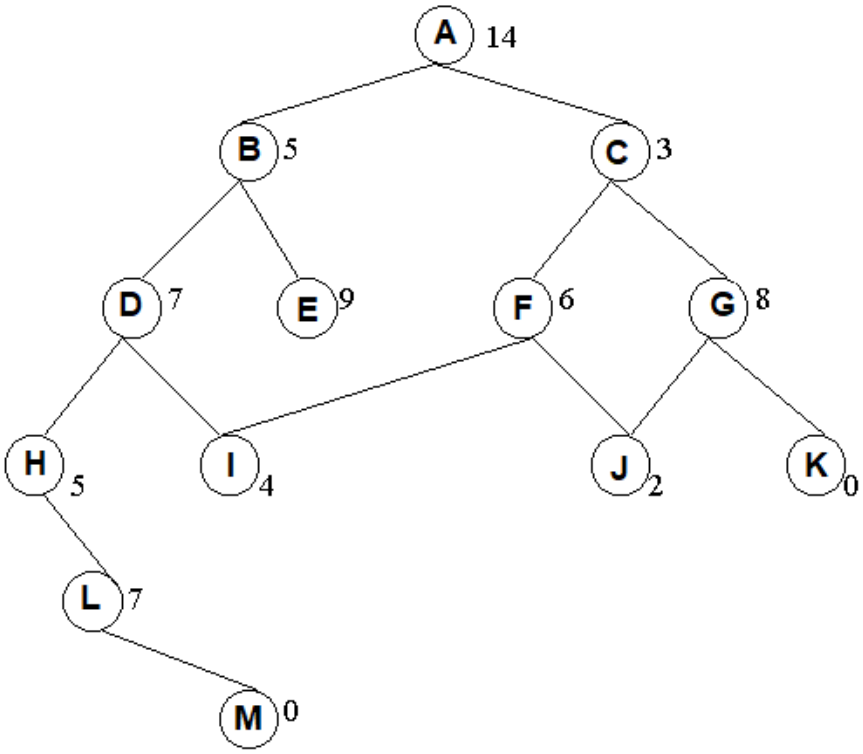
- n representa lo cercano que esta el estado con el que se ha aplicado la función de evaluación de la solución final.
- Es muy importante mantener un equilibrio entre la eficiencia de la función y su complejidad.
- La función de evaluación heurística (fev) no es más que una **estimación de la distancia real a la meta**.
- La fev permitirá guiar la búsqueda hacia aquellos camino que se supone son más prometedores.

3.2 Primero el mejor

- Los nodos se ordenan de tal manera que se expande el nodo de mejor valor de la función heurística ***f*** .
- ***f(n)*** da un valor numérico que indica una medida de lo prometedor que es el nodo para ser expandido.
- Esta función se utiliza para ***ordenar*** la lista ***ABIERTA***, de modo que aquellos nodos más prometedores estarán al principio.
- Esta función puede tener en cuenta o no el costo del mejor camino parcial encontrado en cada momento desde la raíz hasta el nodo considerado.
- Complejidad temporal $O(b^p)$. b/factor de ramificación. p/ profundidad máx
- Complejidad espacial: $O(b^p)$. Depende del problema concreto
- ***No es completa***, en general.

  Por ejemplo, una mala heurística podría hacer que se tomara un camino infinito.
- No es ***minimal*** ya que no garantiza soluciones con el menor número de operadores.
- La heurística podría guiar hacia una solución no minimal

3.2 Primero el mejor: Ejemplo



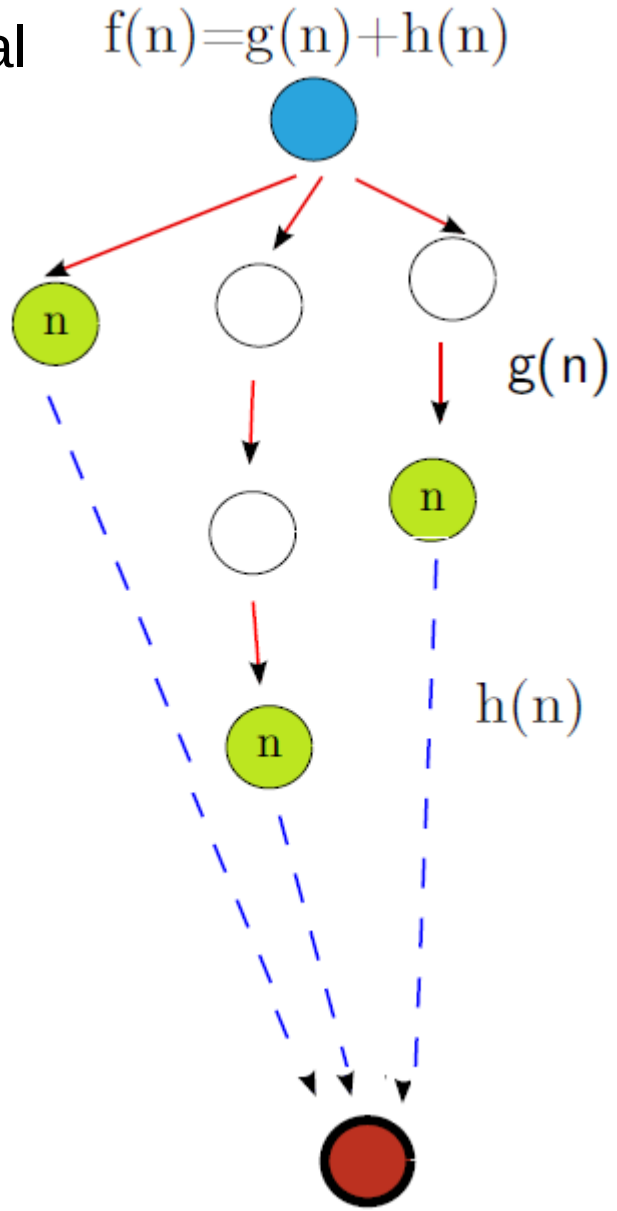
ABIERTA	TABLA_A
Paso 1) A (14)	Vacía
Paso 2) C(3), B(5)	A
Paso 3) B(5), F(6), G(8)	A,C,
Paso 4) F(6), D(7), G(8), E(9)	A,C,B,
Paso 5) J(2), I(4), D(7), G(8), E(9)	A,C,B,F
Paso 6) I(4), D(7), G(8), E(9)	A,C,B,F,J
En este ciclo del algoritmo se encuentra un descendiente (G) del nodo que se está expandiendo (J), ya que estaba en ABIERTA. A pesar de ello no se produce reorientación por ser el anterior camino desde G al nodo raíz de menor costo que el encontrado ahora.	
Paso 7) D(7), G(8), E(9)	A,C,B,F,J,I,
Tenemos un caso análogo al del ciclo anterior.	
Paso 8) H(5), G(8), E(9)	A,C,B,F,J,I,D
Al expandir D se genera I, que ya estaba TABLA_A. El nuevo camino encontrado hasta I no es menos costoso que el anterior; por lo tanto, no hay que redirigir arcos.	
Paso 9) L(7), G(8), E(9)	A,C,B,F,J,I,D,H
Paso 10) G(8),E(9)	A,C,B,F,J,I,D,H,L
Con la generación de M se llega al final del algoritmo. La lista ABIERTA debe mantenerse ordenada a lo largo de todo el proceso, ya que el "siguiente nodo a expandir" es el primero de dicha lista.	
El camino encontrado siguiendo los punteros, que siempre deben señalar al mejor antecesor de un nodo en el grafo, es: A, B, D, H, L, M	

3.3 Algoritmo A*

La función de evaluación tiene dos componentes:

1. coste para ir desde el (un) inicio al nodo actual
2. coste (estimado) para ir desde el nodo actual a una solución

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



- ***f*** es un valor estimado del coste total del camino que pasa por *n*
- ***h*** (heurístico) es un valor estimado de lo que falta para llegar desde *n* al (a un) objetivo
- ***g*** es un coste real (lo gastado por el camino más corto conocido hasta *n*)

La preferencia es siempre del nodo con menor ***f*** , en caso de empate, la preferencia es del nodo con menor ***h***.

3.3 Algoritmo A*: código/Admisibilidad

```
ABIERTA = (inicial);
mientras NoVacia(ABIERTA) hacer
   n = ExtraePrimero(ABIERTA);
   si EsObjetivo(n) entonces
      devolver Camino(inicial, n) y para;
   fin si
   S = Sucesores(n);
   Añade S a la entrada de n en la TABLA_A;
   para cada q de S hacer
      si (q ∈ TABLA_A) entonces
         Rectificar(q, n, Coste(n, q));
         Ordenar(ABIERTA); {si es preciso}
      si no
         pone q en la TABLA_A con
               Anterior(q) = n,
               g(q) = g(n) + Coste(n, q),
               h(q) = Heurístico(q);
         ABIERTA = Mezclar(q, ABIERTA);
      fin si
   fin para
fin mientras
devolver "no solución";
```

Para que A* sea admisible la función heurística h debe ser una estimación optimista del valor de h* para todos los nodos.

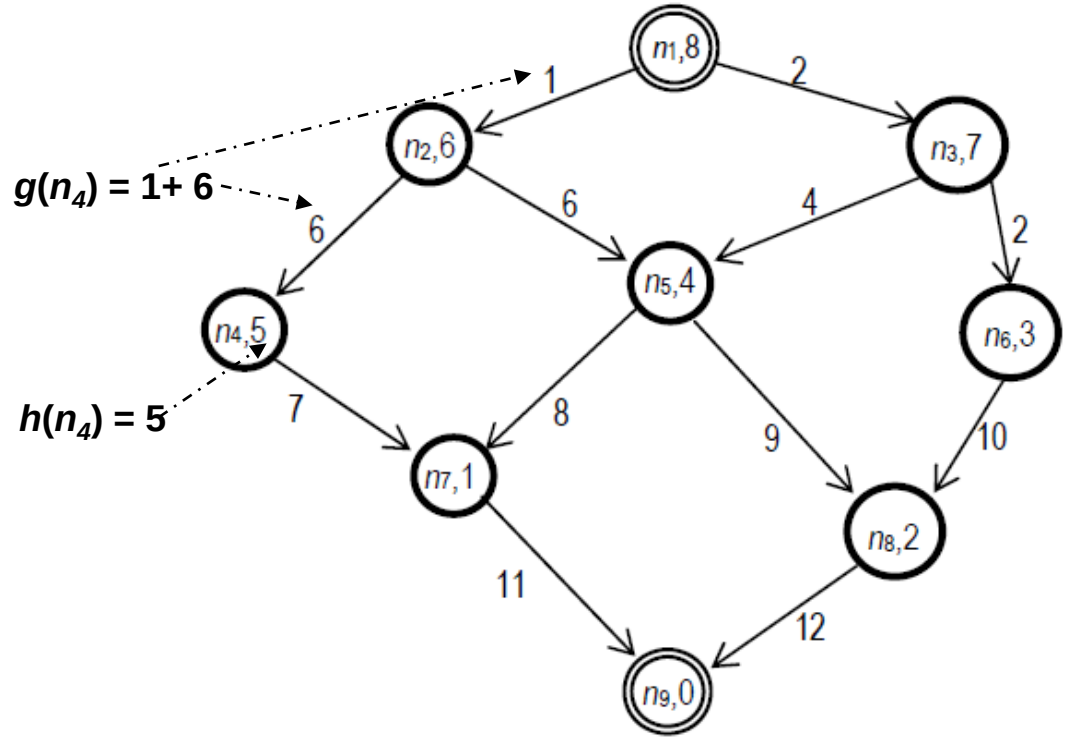
Definición Una función heurística h es admisible si $h(n) \leq h^*(n)$ $\forall n$.

A partir de aquí se supone que la función h es admisible y se probará que el algoritmo A* que la utiliza es admisible.

Teorema 2.3.2. A* es admisible.

3.3 Algoritmo A*: Ejemplo

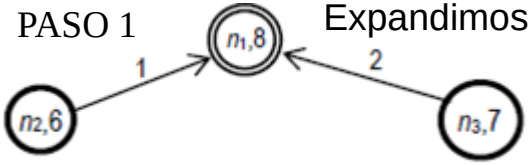
$f(n_4) = g(n_4)+h(n_4)$



PASO 0

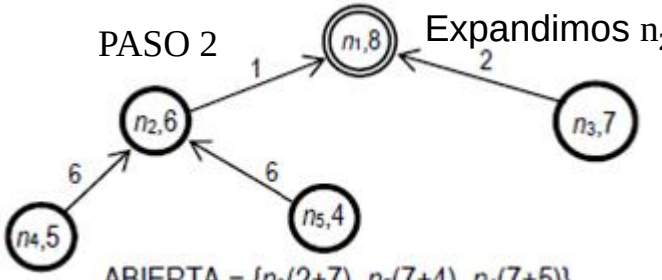
ABIERTA = {n_1(0+8)}

PASO 1 Expandimos n_1



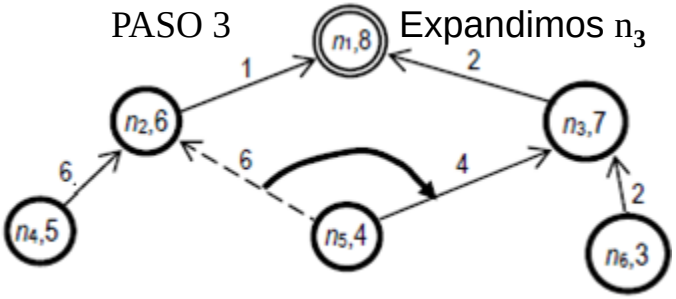
ABIERTA = {n_2(1+6), n_3(2+7)}

PASO 2 Expandimos n_2



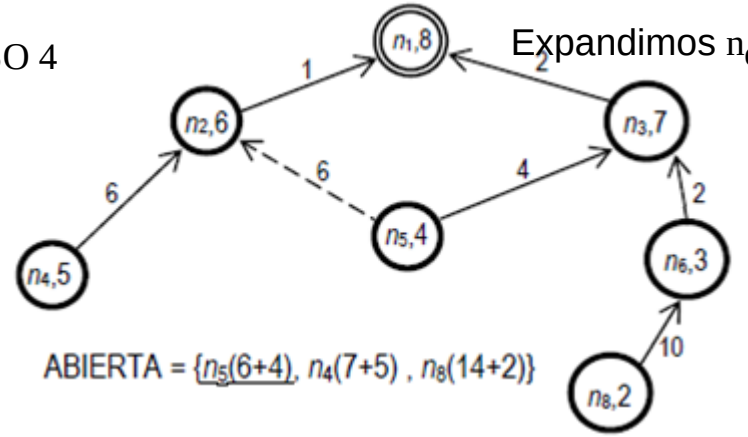
ABIERTA = {n_3(2+7), n_5(7+4), n_4(7+5)}

PASO 3 Expandimos n_3



ABIERTA = {n_6(4+3), n_5(6+4), n_4(7+5)}

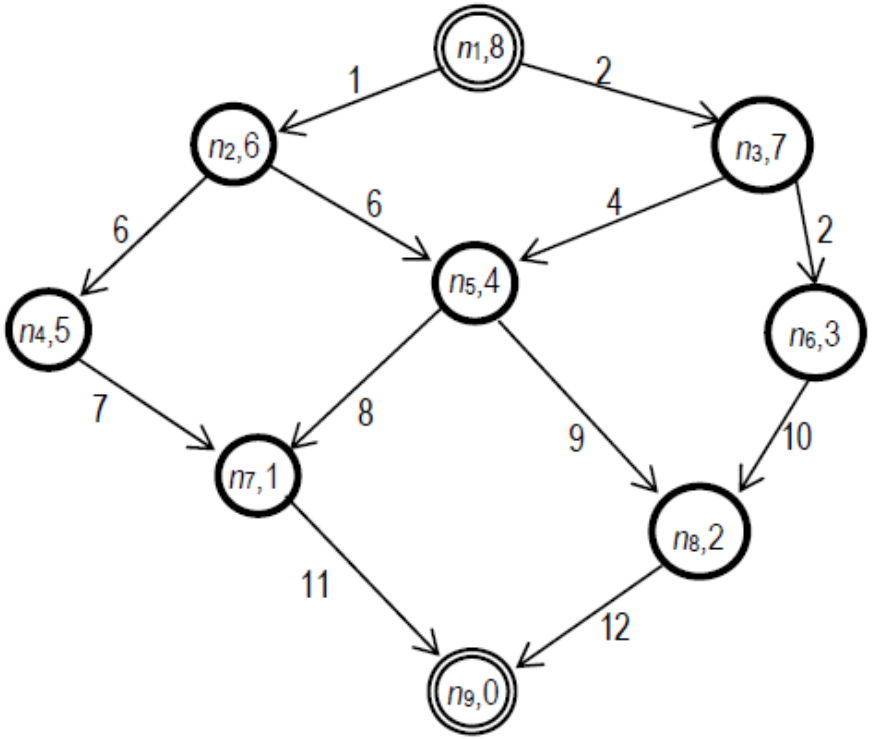
PASO 4 Expandimos n_6



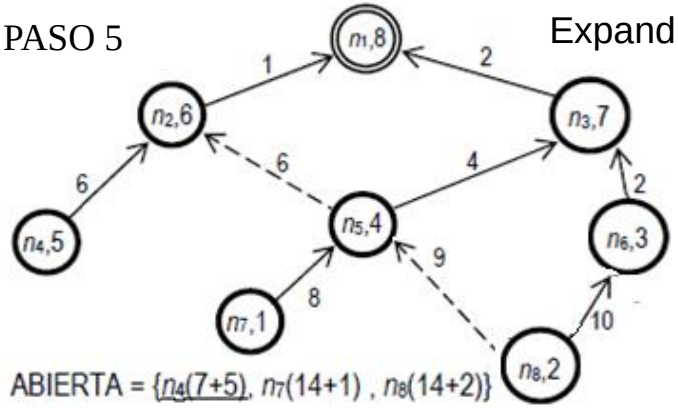
ABIERTA = {n_5(6+4), n_4(7+5) , n_8(14+2)}

El mejor camino desde n_5 al nodo inicial lo marca su padre n_3 (coste 4+2=6) y no n_2 (coste 6+1=7). El arco de n_5 a n_3 se marca con trazo **continuo** y de n_5 a n_2 con trazo **discontinuo**

3.3 Algoritmo A*: Ejemplo

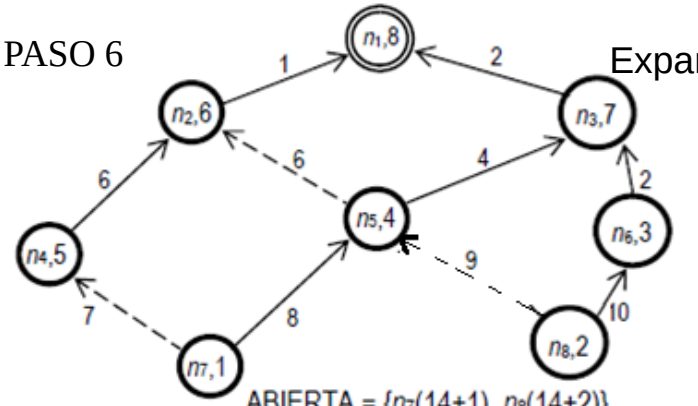


PASO 5 — Expandimos n_5



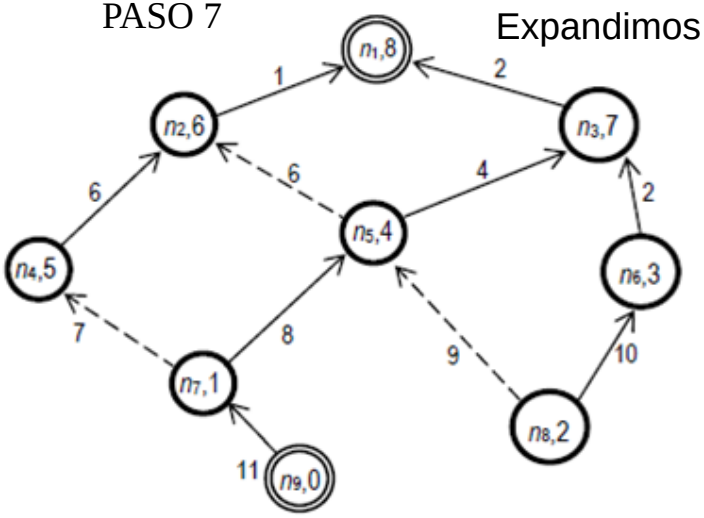
ABIERTA = {n_4(7+5), n_7(14+1) , n_8(14+2)}

PASO 6 — Expandimos n_4



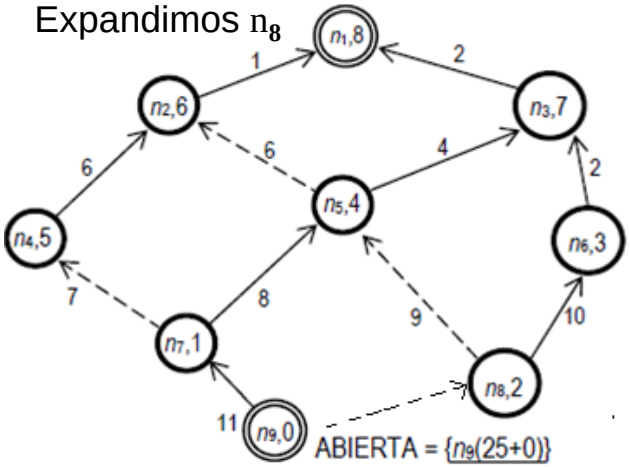
ABIERTA = {n_7(14+1), n_8(14+2)}

PASO 7 — Expandimos n_7



ABIERTA = {n_8(14+2), n_9(25+0)}

PASO 8 — Expandimos n_8

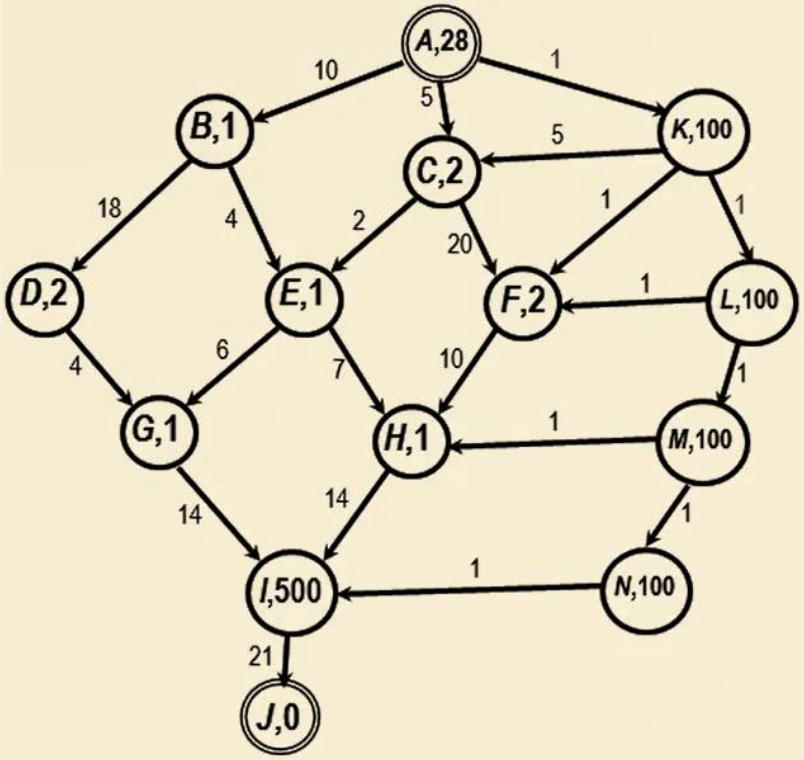


ABIERTA = {n_9(25+0)}

PASO 9

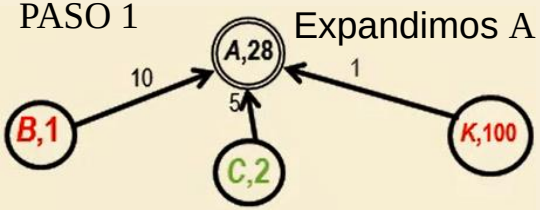
Intentamos expandir n_9 y alcanzamos una meta, con lo que el algoritmo termina. El camino solución encontrado es: $n_1 \rightarrow n_3 \rightarrow n_5 \rightarrow n_7 \rightarrow n_9$, cuyo **coste es 25**.

3.3 Algoritmo A*: Ejemplo2



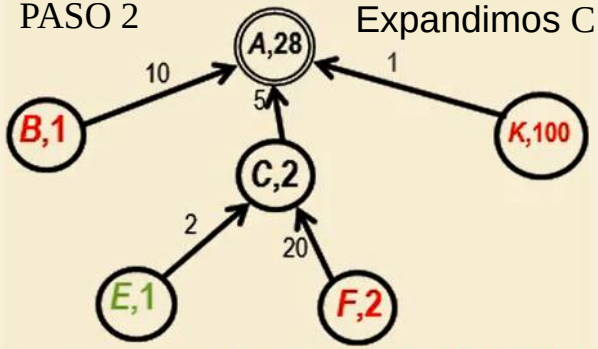
PASO 0  ABIERTA = {A(0+28)}

PASO 1 Expandimos A



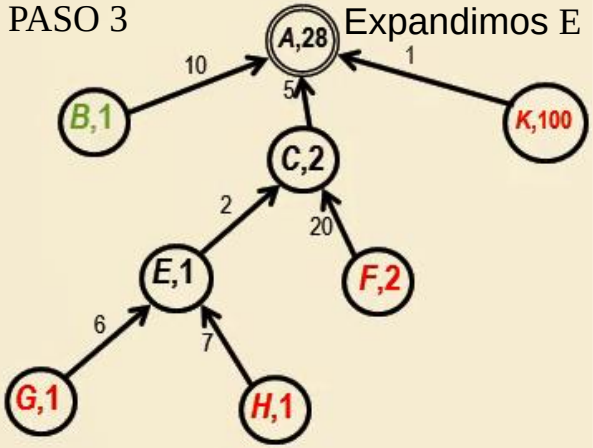
ABIERTA = {C(5+2), B(10+1), K(1+100)}

PASO 2 Expandimos C



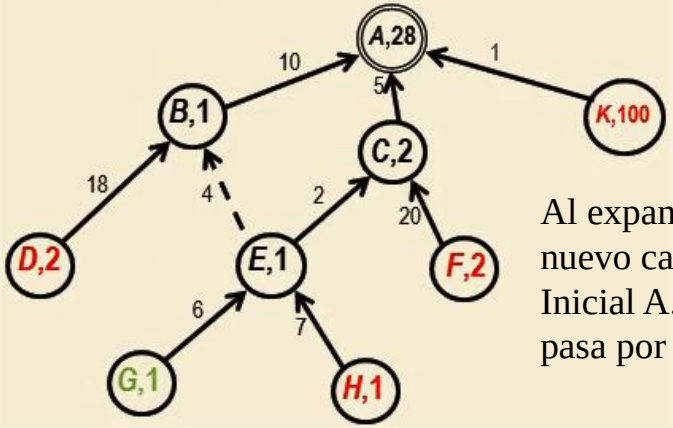
ABIERTA = {E(7+1), B(10+1), F(25+2), K(1+100)}

PASO 3 Expandimos E



ABIERTA = {B(10+1), G(13+1), H(14+1), F(25+2), K(1+100)}

PASO 4 Expandimos B

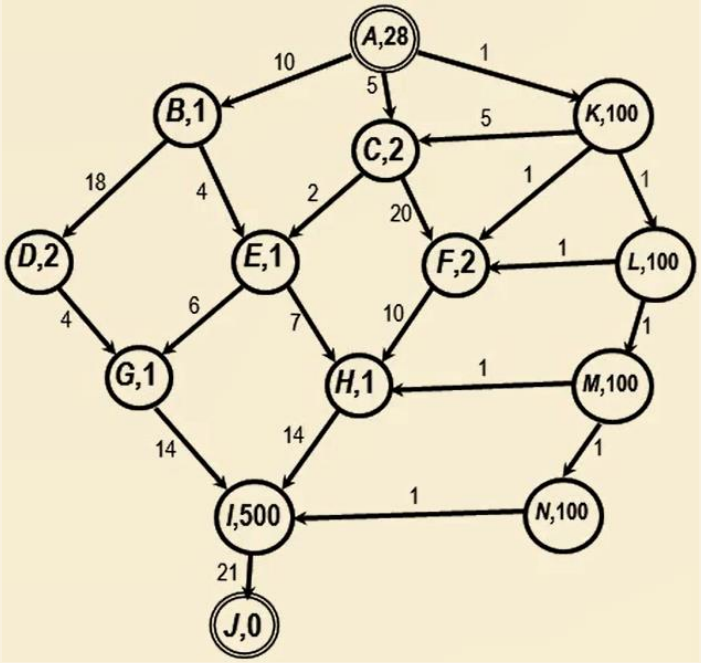


Al expandir B, encontramos un nuevo camino del nodo E al nodo Inicial A. El mejor camino de E a A pasa por C y no por B.

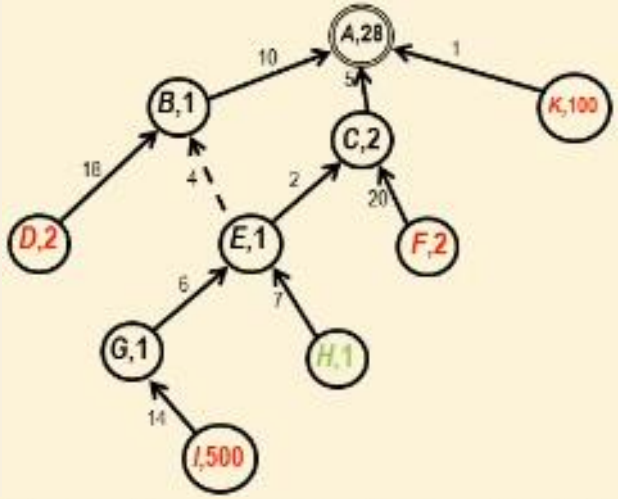
ABIERTA = {G(13+1), H(14+1), F(25+2), D(28+2), K(1+100)}

3.3 Algoritmo A*: Ejemplo2

A-star.mp4

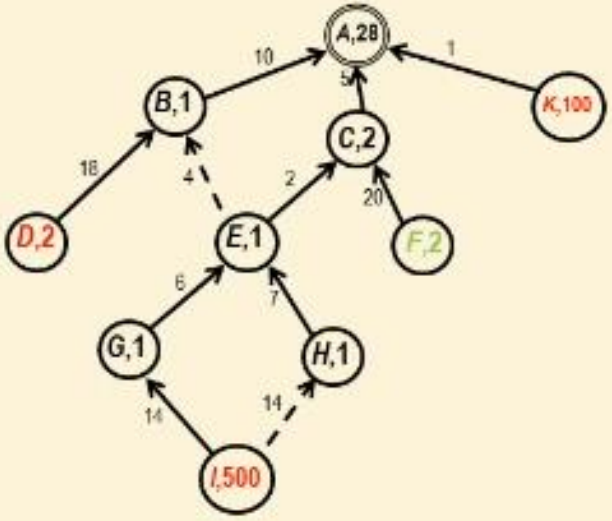


PASO 5 Expandimos G



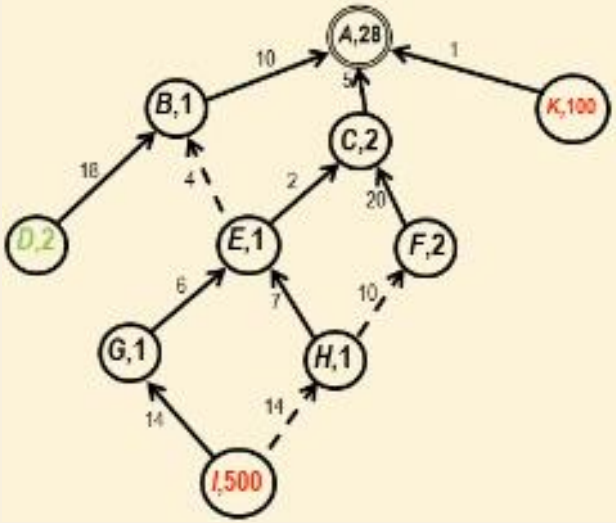
ABIERTA = {*H*(14+1), *F*(25+2), *D*(28+2), *K*(1+100), *I*(27+500)}

PASO 6 Expandimos H



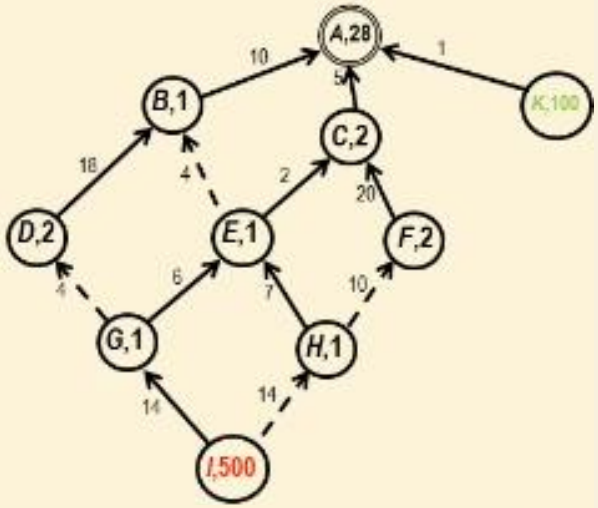
ABIERTA = {*F*(25+2), *D*(28+2), *K*(1+100), *I*(27+500)}

PASO 7 Expandimos F



ABIERTA = {*D*(28+2), *K*(1+100), *I*(27+500)}

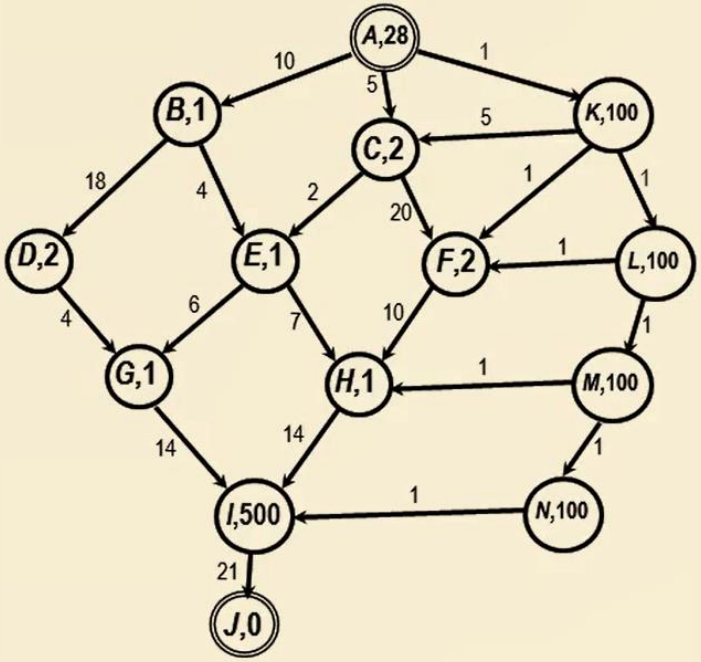
PASO 8 Expandimos D



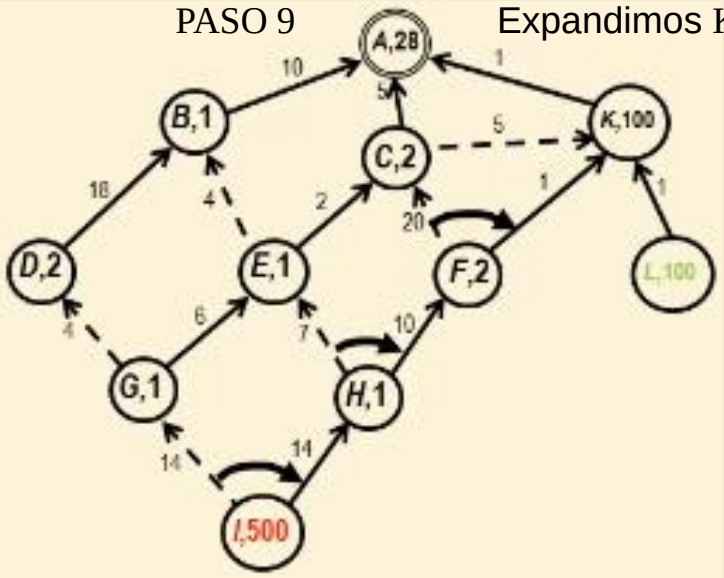
ABIERTA = {*K*(1+100), *I*(27+500)}

3.3 Algoritmo A*: Ejemplo2

A-star.mp4

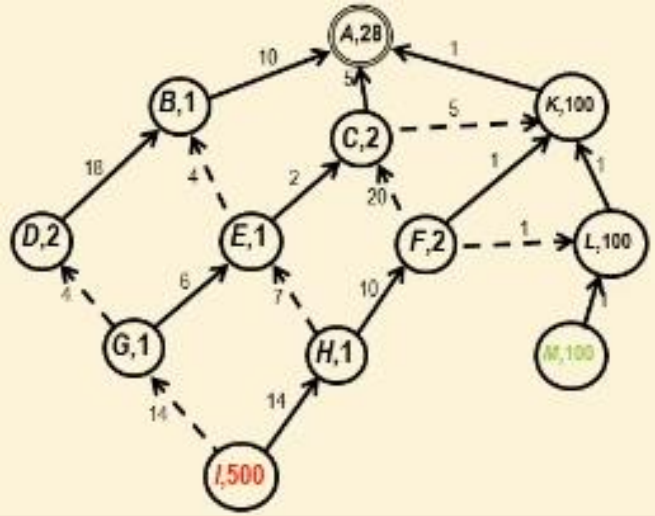


PASO 9 Expandimos K



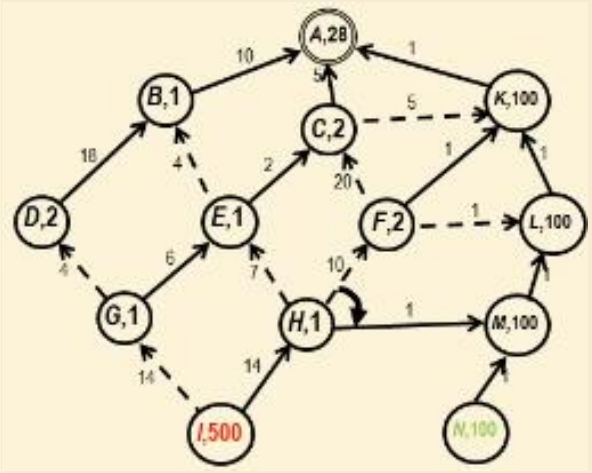
ABIERTA = {L(2+100), I(26+500)}

PASO 10 Expandimos L



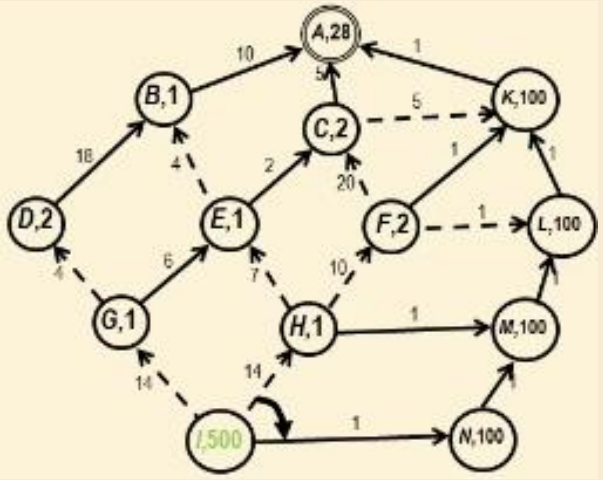
ABIERTA = {M(3+100), I(26+500)}

PASO 11 Expandimos M



ABIERTA = {N(4+100), I(18+500)}

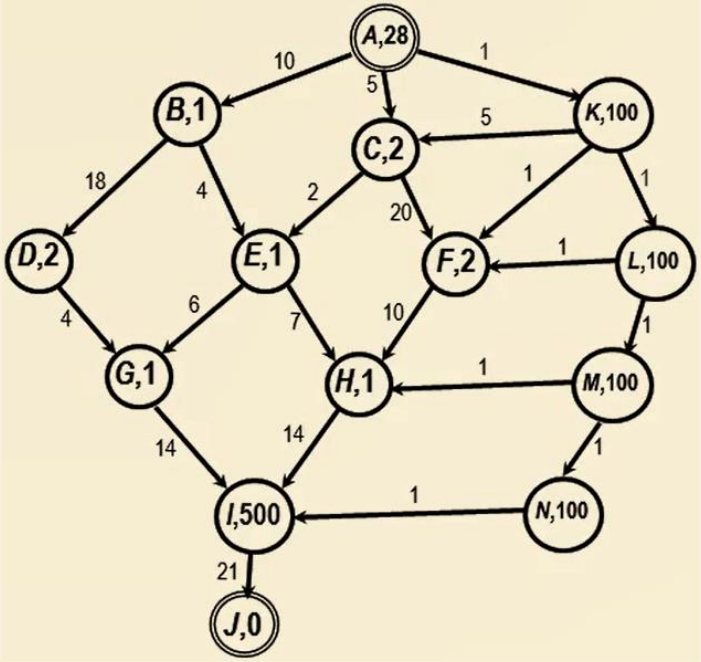
PASO 12 Expandimos N



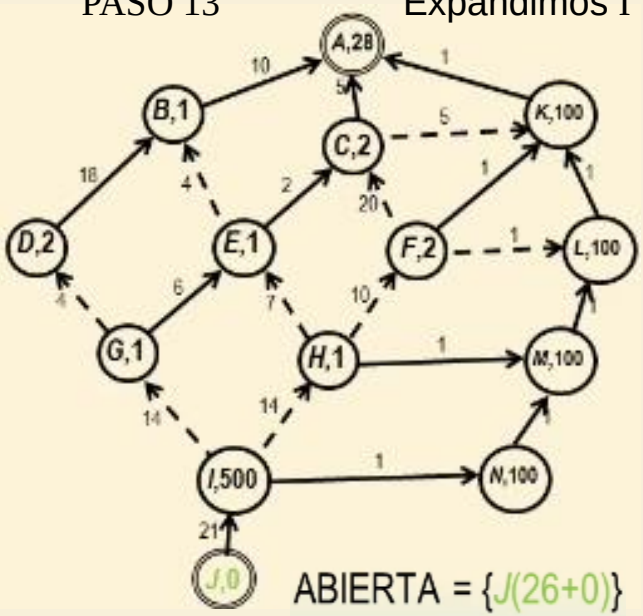
ABIERTA = {I(5+500)}

3.3 Algoritmo A*: Ejemplo2

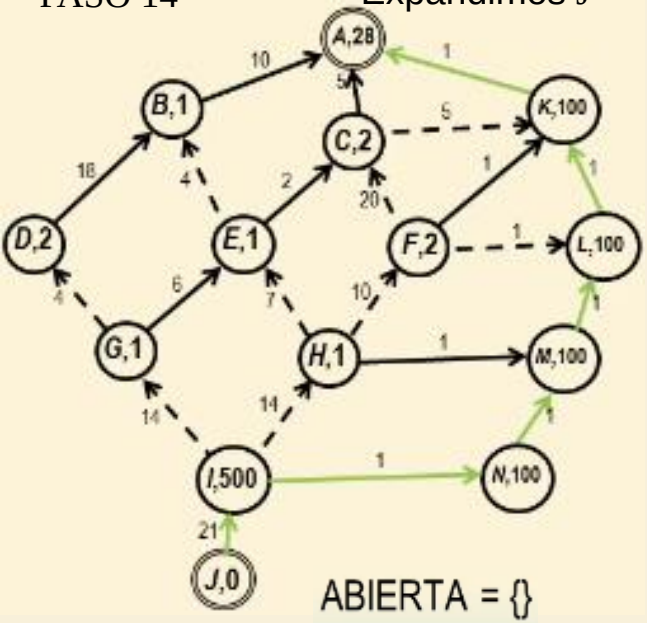
A-star.mp4



PASO 13 Expandimos I



PASO 14 Expandimos J



PASO 11 Expandimos M

PASO 12 Expandimos N

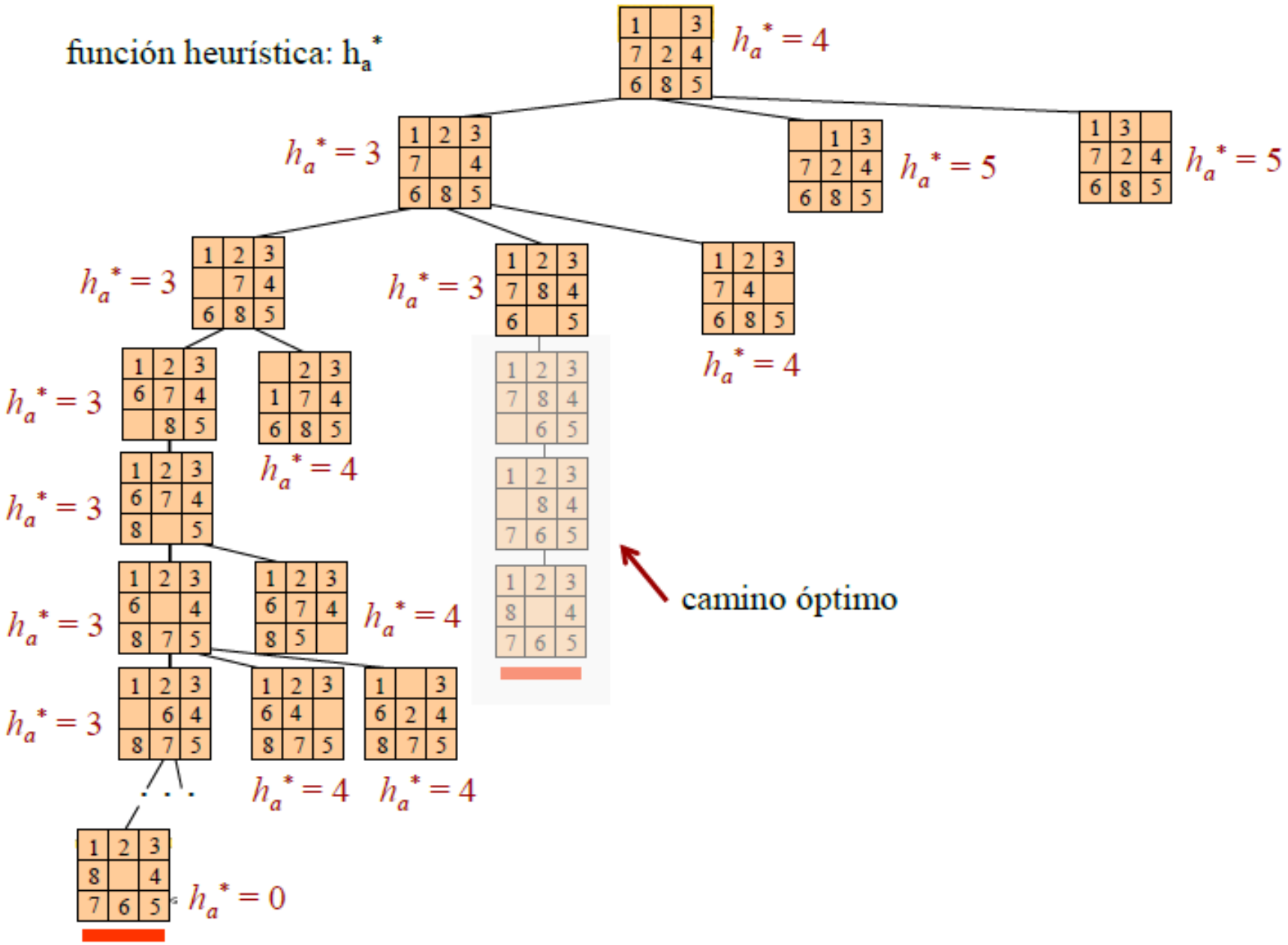
Camino solución → **A K L M N J**

Coste → 26

3.4 Algoritmos voraces

- Constituyen una alternativa a la búsqueda exhaustiva.
- Los algoritmos voraces se basan en la idea de que la toma decisiones de se hace de forma irrevocable, de modo que nunca considera otra alternativa.
- Los algoritmos voraces **no son admisibles**, y en general **tampoco son completos**.
- Pero por otra parte resultan **extremadamente eficientes**, por lo que son muy adecuados por ejemplo para los sistemas de tiempo real
- El diseño de un algoritmo voraz para la búsqueda en espacios de estados se obtiene a partir de un algoritmo A*, si en cada expansión se prescinde de todos los sucesores salvo del más prometedor de acuerdo con el heurístico h, se tiene un algoritmo voraz.

Búsqueda voraz (evitando ciclos simples): Ejemplo 1



3.5. Algoritmos de ramificación y poda

- Es el método más empleado en la resolución de problemas de optimización combinatoria.
- Cada estado se interpreta como un subconjunto de soluciones del problema original,
- El espacio de búsqueda tiene estructura de árbol.
- Los algoritmos de ramificación y poda tienen cuatro componentes fundamentales:
  - Un esquema de ramificación,
  - Un método de cálculo de cotas inferiores (Lower Bounds),
  - Un método de cálculo de cotas superiores (Upper Bounds)
  - Una estrategia de control.

Esquema de ramificación.

- Cada nodo representa un problema y dado un nodo n, Y(n) representa al conjunto de todas las soluciones de n.
- Dado un nodo n el algoritmo de cálculo de cotas inferiores produce una cota inferior de la mejor solución de Y (n) (el valor de esta cota inferior es análogo al valor f(n) = g(n) + h(n) en A* si tenemos un heurístico admisible).

3.5. Algoritmos de ramificación y poda: Algoritmo

$ABIERTA = inicial$;
$UB = \infty$;
$s = \oslash$;
$LB =$ cota inferior de $inicial$;
mientras ($NoVacia(ABIERTA)$ y $(LB < UB)$ y (No $CriterioTerminación$))) **hacer**
  Extrae un elemento n de $ABIERTA$;
  $s_n =$ solución de $Y(n)$ obtenida mediante un algoritmo heurístico;
  si $(Coste(s_n) < UB)$ **entonces**
    $s = s_n$;
    $UB =$ valor de s_n;
  fin si
  Calcula los sucesores de n, $q_1, \ldots, q_p$, ordenados con algún criterio, de modo que $Y(n) = Y(q_1) \cup \cdots \cup Y(q_p)$, siendo esta unión disjunta;
  Calcula cotas inferiores LB_i de q_i, $1 \leq i \leq p$;
  para cada i de 1 a p **hacer**
    si $(LB_i < UB)$ **entonces**
      si $(|Y(q_i)| = 1)$ **entonces**
        $UB = LB_i$;
        $s =$ la solución de $Y(q_i)$;
      si no
        Añade q_i a $ABIERTA$;
      fin si
    fin si
  fin para
  $LB =$ mín $(LB_q, q \in ABIERTA)$;
fin mientras
devolver s, UB, LB;

3.5. Algoritmos de ramificación y poda: poda αβ

El método de **poda αβ** se encarga de llevar una anotación para saber cuándo se puede realizar una "poda" en el árbol de búsqueda.

Usa en cada llamada recursiva a un nodo hijo, dos valores **(α, β**) de manera que:

- **alfa (α)** marque la **cota inferior** de los valores que se van a ir buscando en la parte del árbol que queda por explorar,
- **beta (β)** la **cota superior** de esos mismos valores.

Ventajas: Ahorro de la exploración de caminos inútiles. Abandona el estudio de caminos que no conducen a soluciones mejores que las ya encontradas.

Reglas:

1.Cuando nos encontramos en un nodo **MAX** modificamos el valor de **α**.

α = max(α, $f_{αβ}$). Si α >= β devolver β.

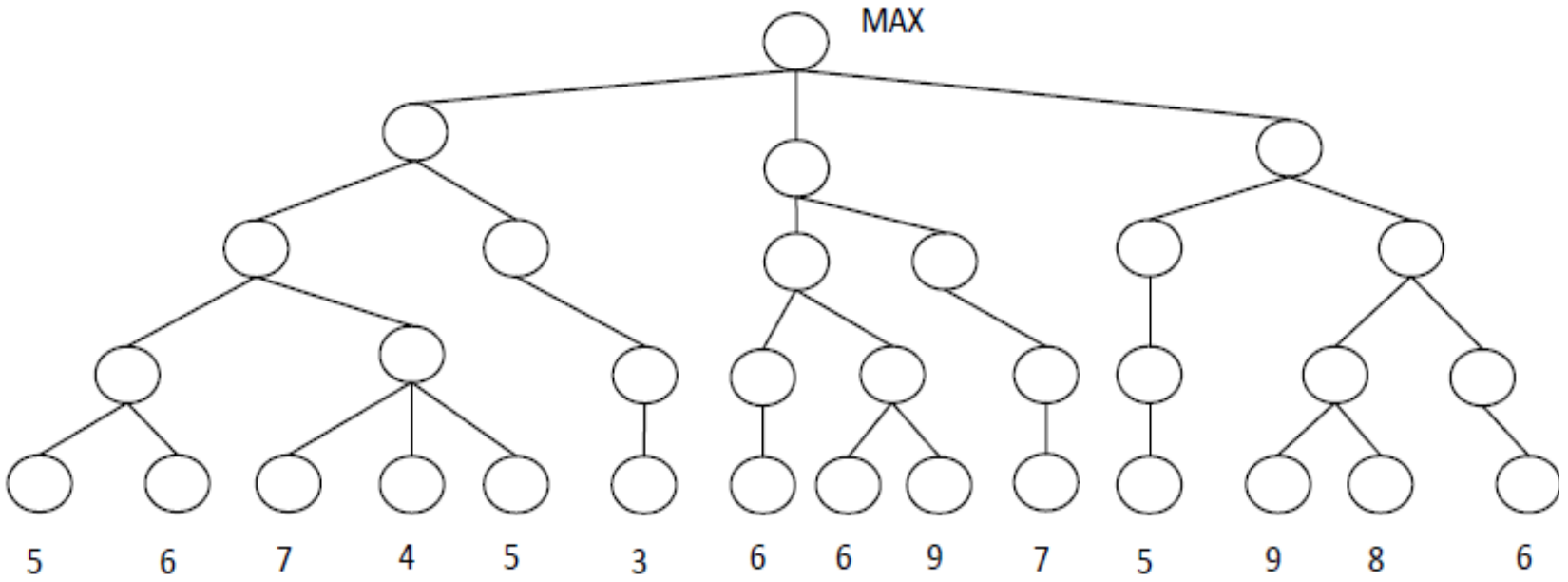
2.Cuando nos encontramos en un nodo **MIN** modificamos el valor de **β**.

β = min (β, $f_{αβ}$). Si α >= β devolver α.

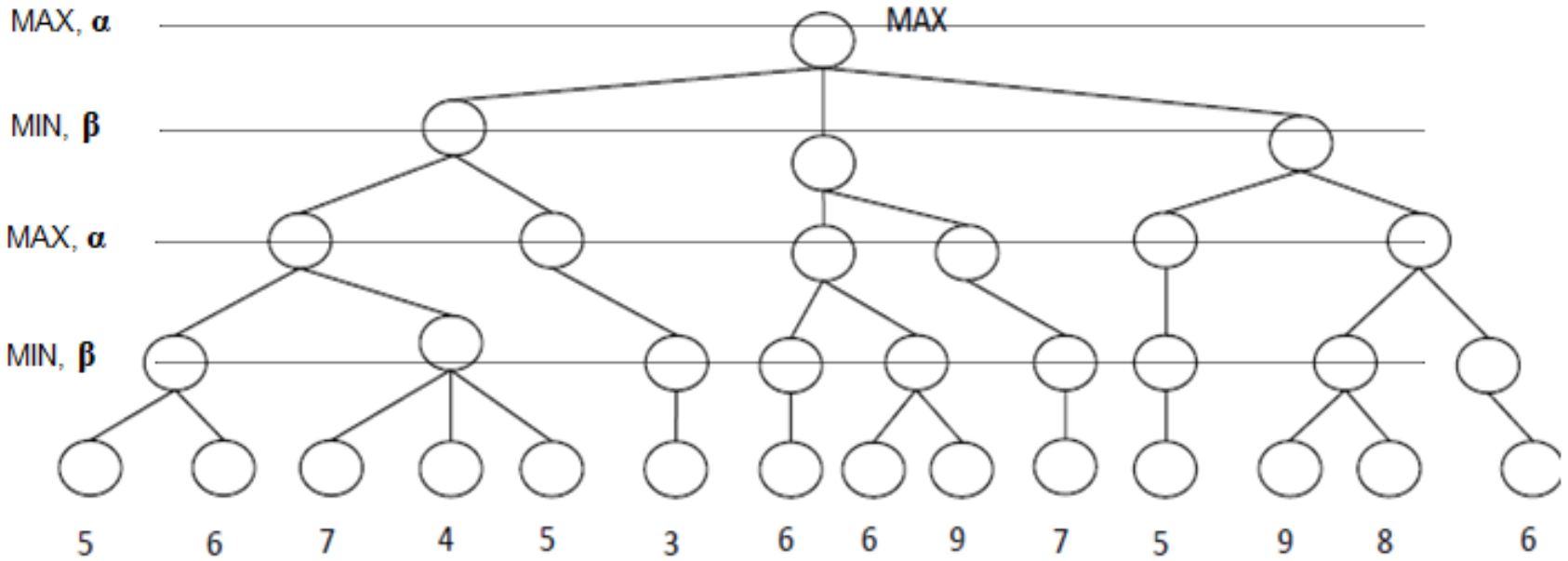
3.Si **α>=β → se poda**, lógicamente no tiene sentido seguir con la búsqueda, realizándose una poda alfa (**α**) si estamos en un nodo **MAX** o beta (**β**) si estamos en un nodo **MIN**.

4.En la primera llamada los valores para α y β son: **α= -∞, β=** ∞ (que se irán heredando hasta llegar a una hoja del árbol). **(-∞,∞)**

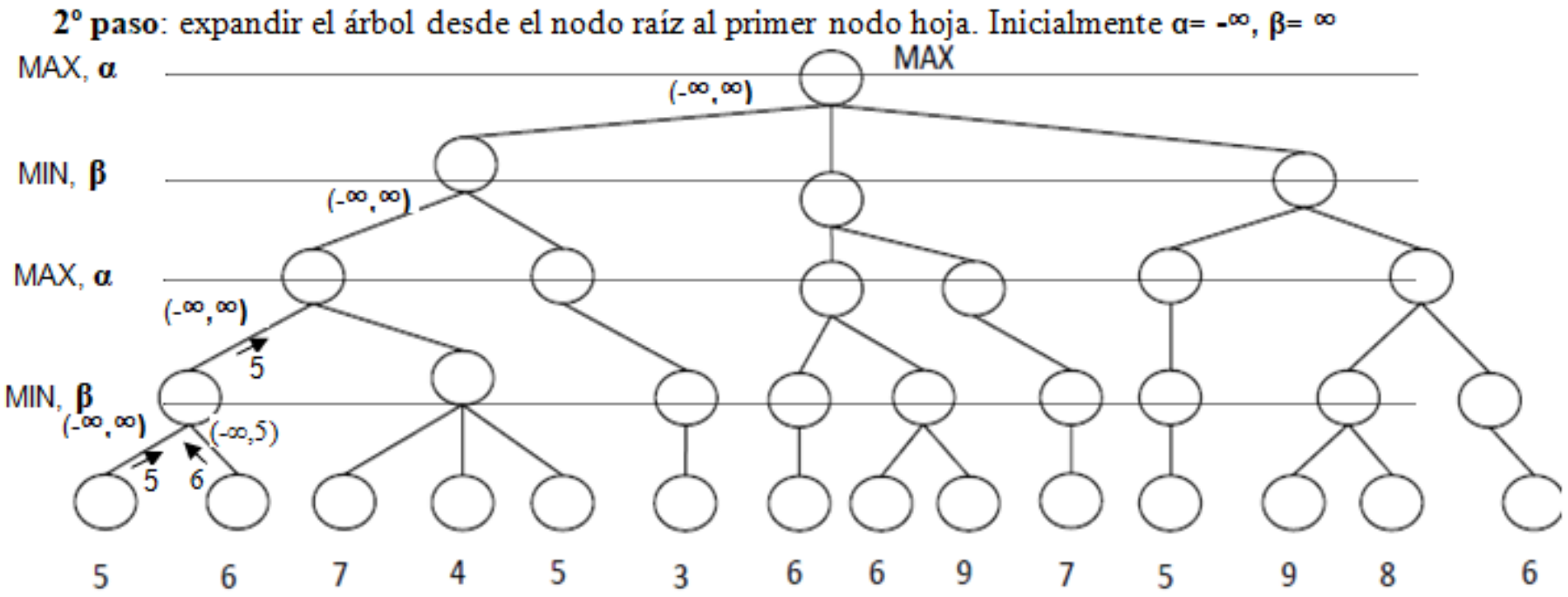
3.5. Algoritmos de ramificación y poda: ejemplo poda αβ



1º paso: marcar los niveles de MAX y MIN



3.5. Algoritmos de ramificación y poda: ejemplo poda αβ



Cuando se llega al primer nodo hijo cuyo valor de estimación de lo comprometedor que es este nodo (5) se actualiza el valor del padre, que en este caso, es un nodo MIN y por lo tanto se actualiza el valor de **β** = min (β, $f_{\alpha\beta}$) = min (∞, 5) = 5.

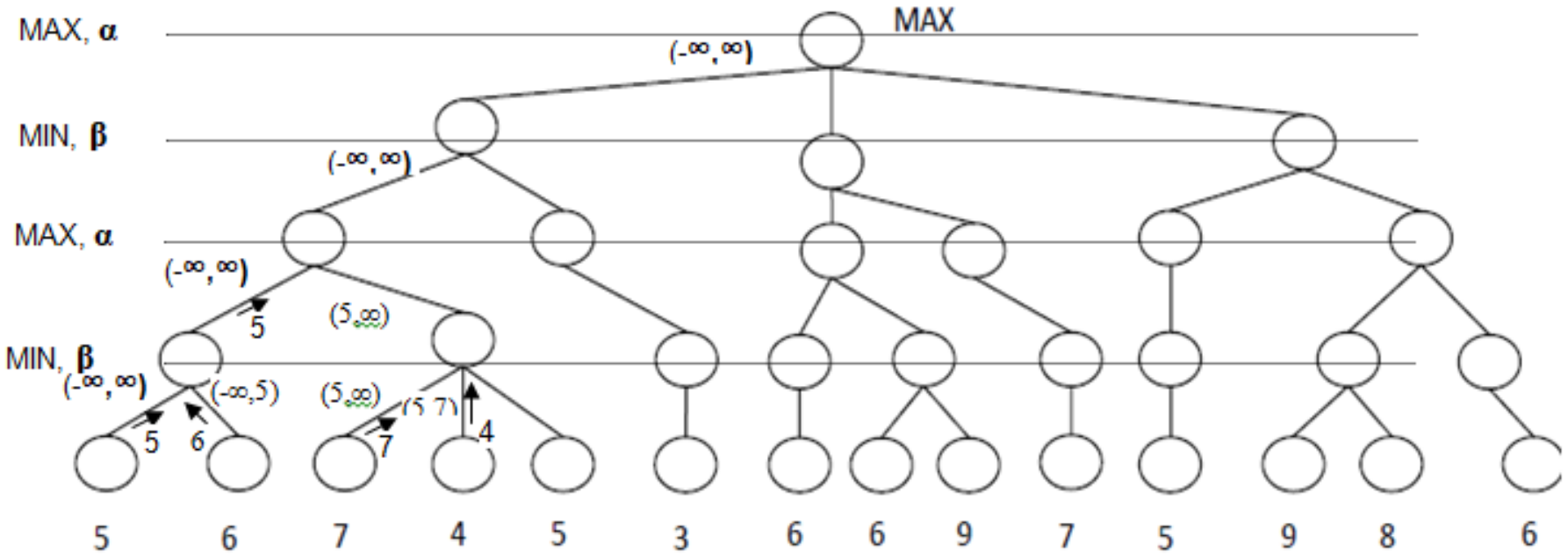
El valor del padre se actualiza a (-∞, 5).

Se repite el proceso con el siguiente hijo a la derecha (valor= 6). **β** = min (5, 6) = 5.

Al no haber más hijos el nodo MIN devuelve el valor de **β** = 5.

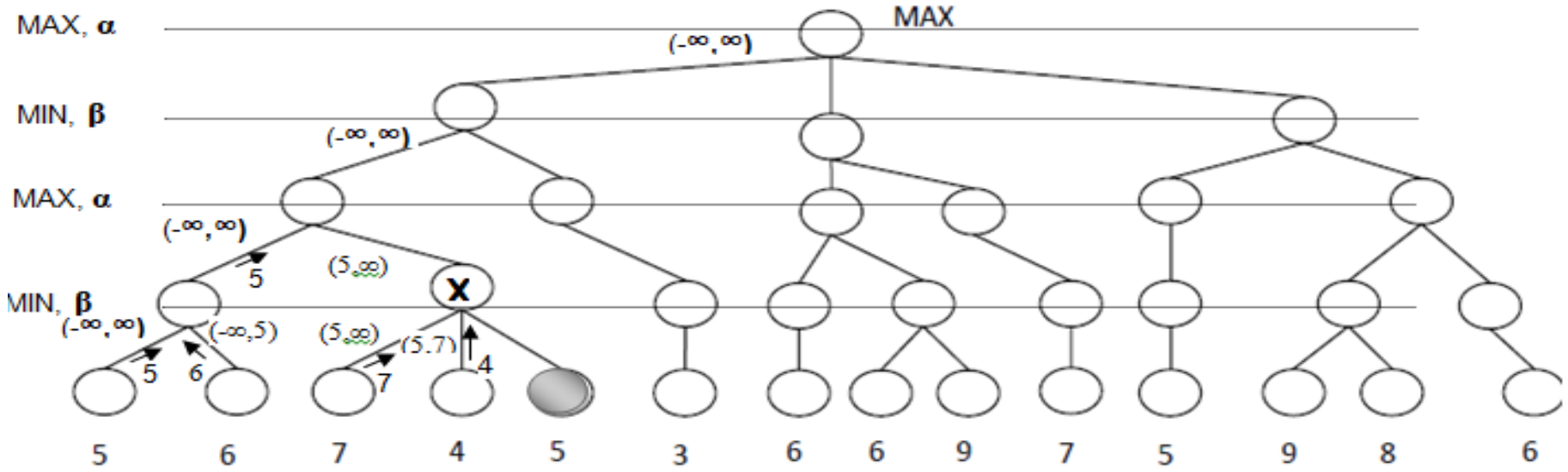
NODO MAX: Se actualiza el valor en el nodo MAX, α= max(α, $f_{\alpha\beta}$) =max (-∞, 5) = 5

3.5. Algoritmos de ramificación y poda: ejemplo poda αβ

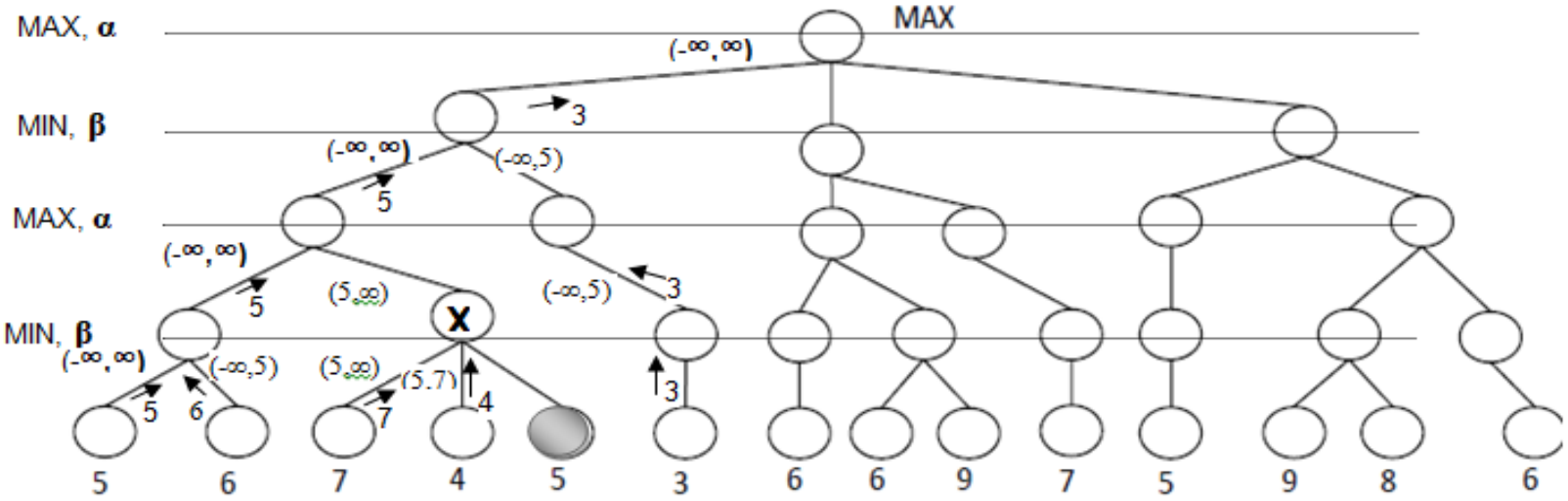


Cuando se llega al nodo hijo cuyo valor de estimación de lo comprometedor que es este nodo (7) se actualiza el valor del padre, que en este caso, es un nodo MIN y por lo tanto se actualiza el valor de **β** = min (β, $f_{\alpha\beta}$) = min (∞**,** 7) = 7.
El valor del padre se actualiza a (5, 7).
Repetimos el proceso con el siguiente hijo a la derecha (valor= 4).
β = min (7, 4) = 4.
Como α=5 >= β =4, **se poda** en ese nodo y se devuelve el valor de α = 5.
Se observa que el nodo cuyo valor de fev es 5 no se explora (se marca en el árbol como tal)

3.5. Algoritmos de ramificación y poda: ejemplo poda αβ



NODO MIN: Se actualiza el nodo MIN, **β** = min (∞**,** 5) = 5. El valor del padre se actualiza a (5, 5).



Cuando se llega nodo hijo cuyo valor de estimación de lo comprometedor que es este nodo (3) se actualiza el valor del padre, que, es un nodo MIN y se actualiza el valor de **β** = min (β, $f_{\alpha\beta}$) = min (**5,** 3) = 3. El valor del padre se actualiza a (-∞, 3).

3.6. Algoritmos de escalada o máximo gradiente

En la búsqueda por máximo gradiente el criterio de aceptación de la solución vecina S1, frente a la solución actual S, es que S1 sea mejor o igual que S. Previamente en la selección se intenta elegir una solución que mejore a S,

Tiene problemas en las siguientes situaciones:

Óptimos locales: cuando se alcanza uno de estos puntos del espacio de búsqueda, todos los vecinos son peores con lo que la búsqueda finaliza sin encontrar el óptimo global.

Regiones planas: en este caso todos los vecinos tienen el mismo valor que la solución actual y la búsqueda es aleatoria.

Crestas: si hay una cresta con pendientes muy marcadas, puede ser fácil alcanzar la cima de la cresta. Sin embargo, si la pendiente de la cima es muy suave en la dirección del óptimo global, resulta difícil guiar la búsqueda a través de la cima sin desviarse hacia los lados de la cresta; a menos que existan operadores específicos para generar vecinos a través de la cima.

En estos tres casos lo que ocurre es que la búsqueda se queda estancada en un óptimo local, que puede ser una solución razonable o no serlo.

3.7. Búsqueda tabú

- La diferencia esencial de la búsqueda tabú con respecto a los métodos anteriores es que dispone de un mecanismo de memoria adaptativa.
- Este mecanismo se utiliza para evitar la generación de algunos vecinos dependiendo de la historia reciente, o de la frecuencia con la que se realizaron algunas transformaciones para llegar al estado actual.
- Por ejemplo, si la regla de vecindad es simétrica, y se genera ***x*** como vecino de ***y***, suele ser buena idea recordarlo y no generar ***y*** inmediatamente a partir de ***x***.

3.7. Búsqueda tabú: ejemplo