

3. ESTRATEGIAS DE BÚSQUEDA INFORMADAS

IA 3.2 - Programación III

1° C - 2023

Lic. Mauro Lucci



Repaso

Estrategias de búsqueda no informadas:

1. Breadth First Search (BFS).
2. Depth First Search (DFS).
3. Uniform Cost Search (UCS).
4. Iterative Deepening Search (IDS).

No usan información adicional más allá de la definición del problema. Todo lo que hacen es expandir nodos siguiendo algún orden particular y distinguir estados objetivos de no-objetivos.

¿Cómo podemos guiar la búsqueda para encontrar soluciones de forma más eficiente?

Estrategias de búsqueda **informadas**

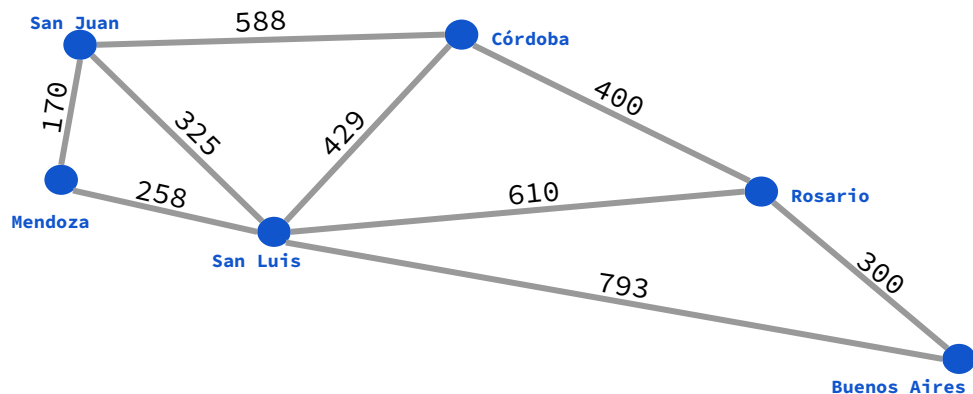
— — —

- Saben si un estado no-objetivo es **más prometedor** que otro y lo expanden antes.
- Usan **conocimiento específico** del problema, más allá de la definición del problema en sí.
- Incorporan una **función heurística** h donde, para cada nodo n ,
 $h(n)$ = costo estimado del camino de menor costo desde el estado en el nodo n a un estado objetivo.
- Si n es un nodo objetivo, entonces $h(n) = 0$.



Ejemplo – Problema de camino más corto

— — —



**Distancia en línea
recta** a San Juan desde:

Buenos Aires	=	997
Rosario	=	747
Córdoba	=	417
San Luis	=	284
Mendoza	=	148
San Juan	=	0

Búsqueda avara primero el mejor

Greedy Best-First Search
(GBFS)

Intenta encontrar
rápidamente una solución.

Los nodos del árbol de
búsqueda se expanden según
su cercanía a un nodo
objetivo.

Dada una función heurística
 h , se expande siempre el
nodo n de la frontera con el
menor costo $h(n)$.

— — —



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1			
2			
3			

997

A:
Bs. As.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	
2			
3			

997

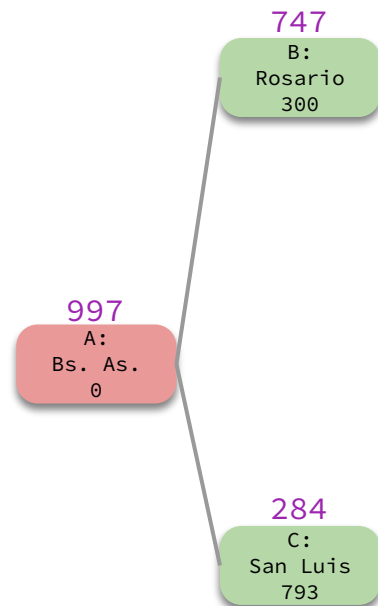
A:
Bs. As.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2			
3			

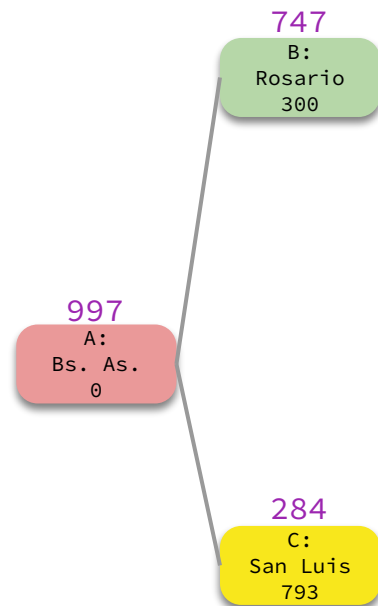




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	
3			

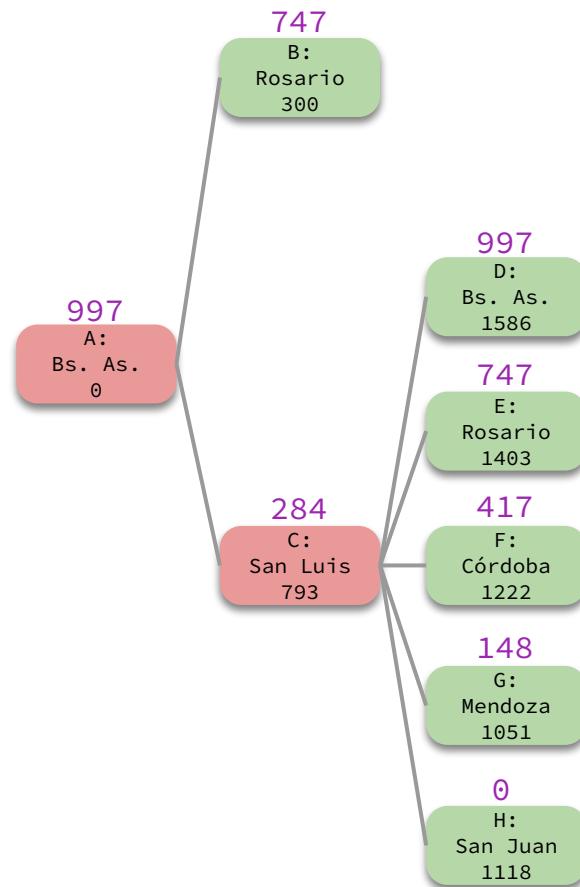




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G,H}
3			

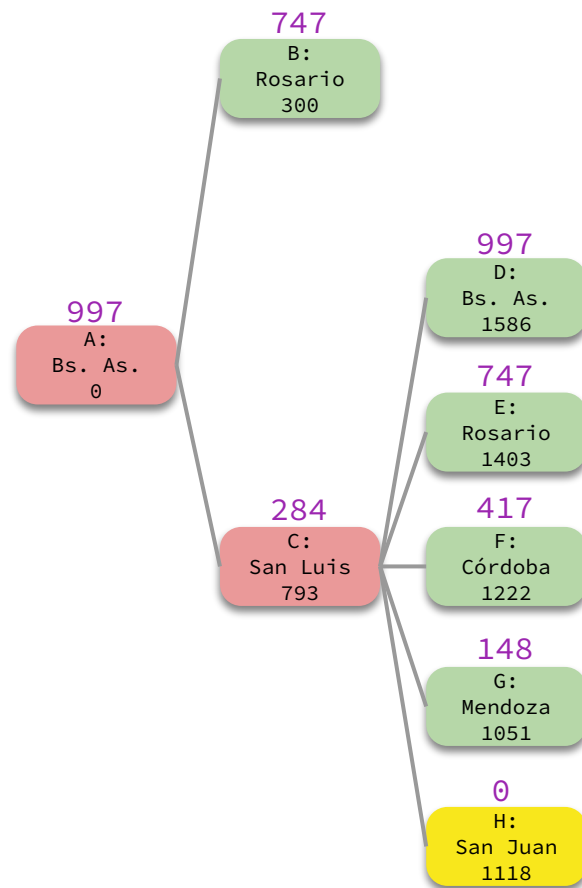




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G,H}
3	H	{B,D,E,F,G}	FIN

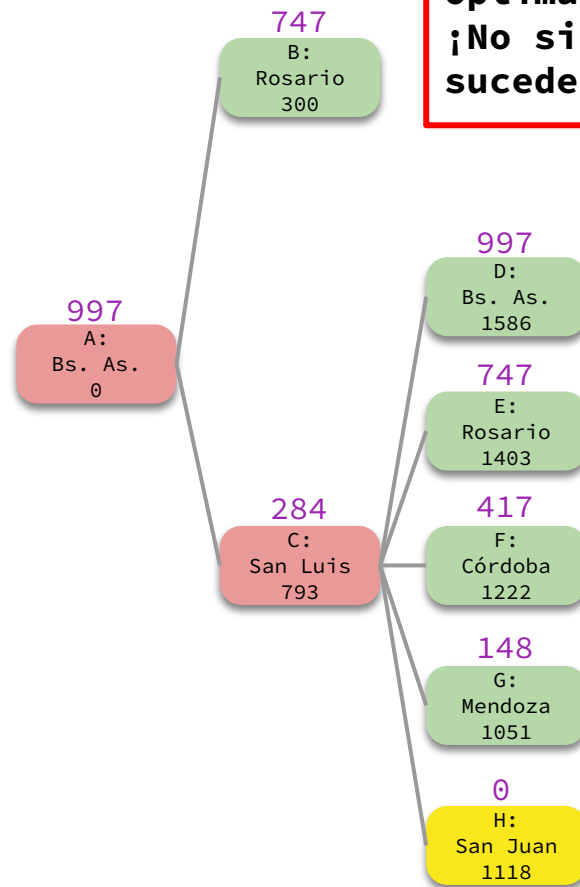




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G,H}
3	H	{B,D,E,F,G}	FIN



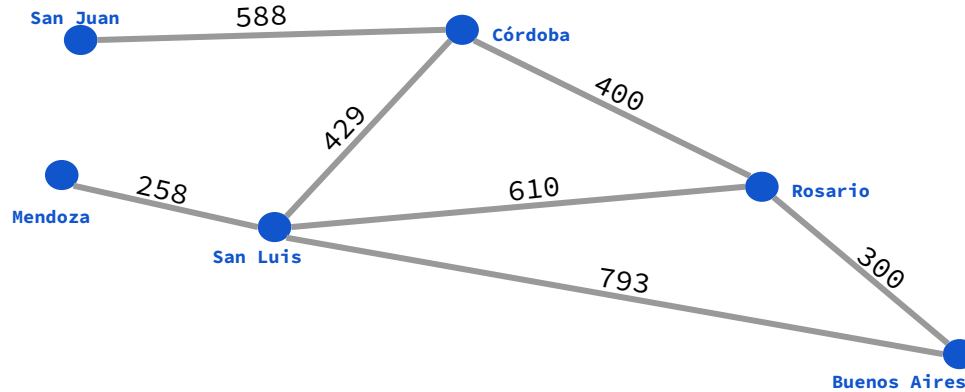
**Encontramos
una solución
óptima.
¡No siempre
sucede!**



Ejemplo – Completitud

— — —

Encontrar el camino más corto de Bs. As. a San Juan en el siguiente mapa:





Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1			
2			
3			
4			

997
A:
Bs. As.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	
2			
3			
4			

997

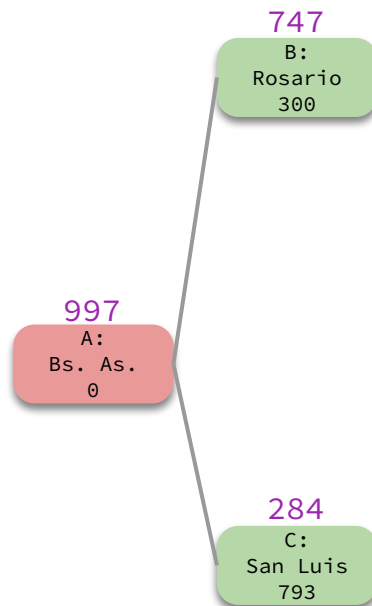
A:
Bs. As.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2			
3			
4			

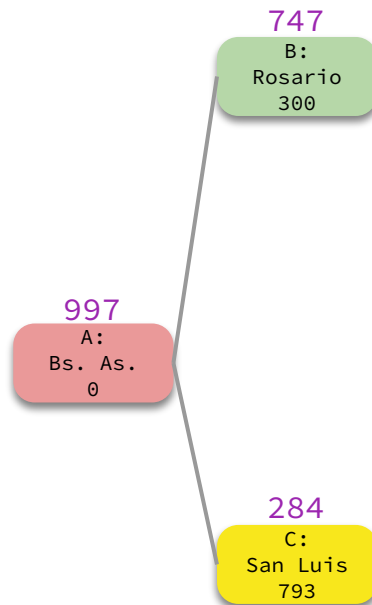




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	
3			
4			

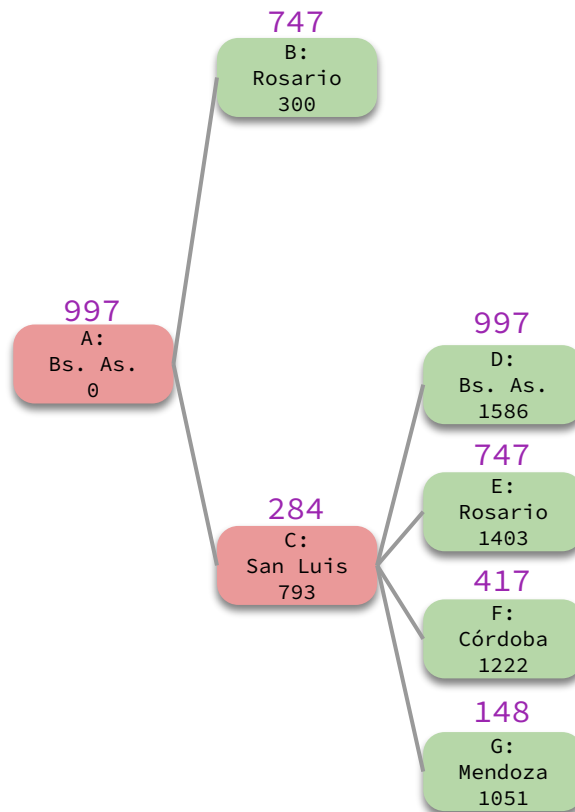




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G}
3			
4			

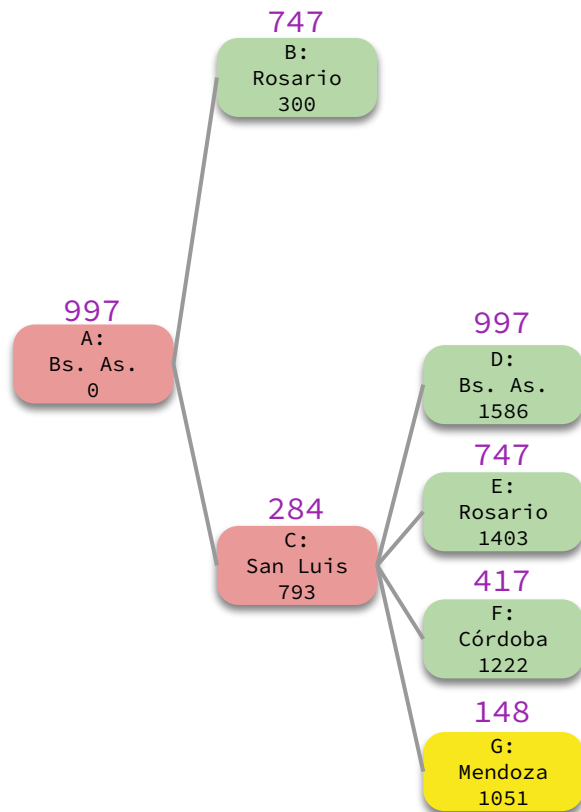




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G}
3	G	{B,D,E,F}	
4			

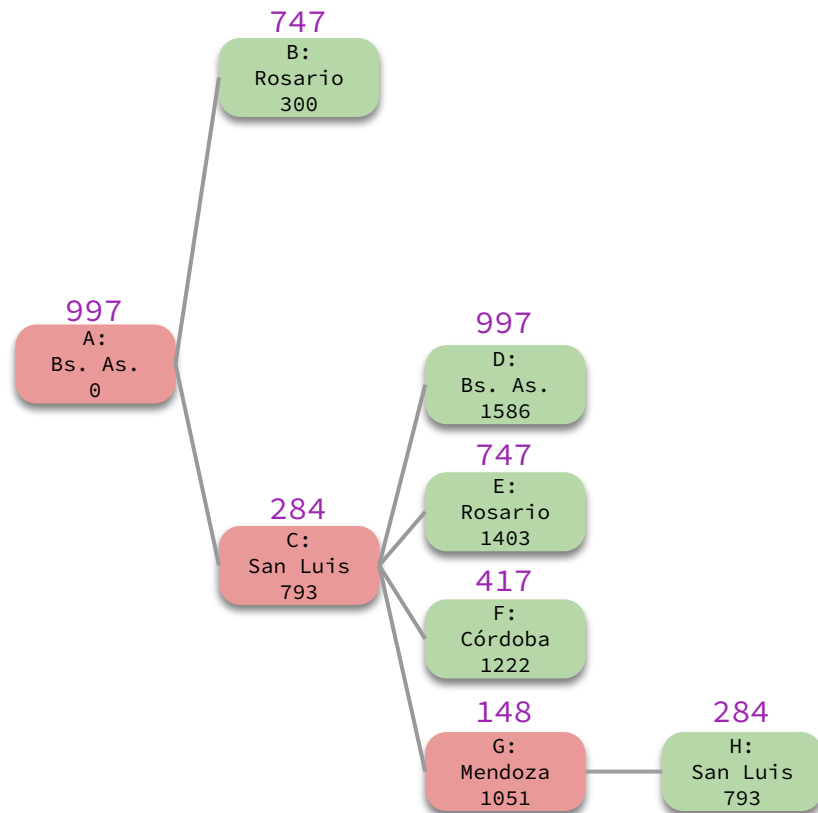




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G}
3	G	{B,D,E,F}	{B,D,E,F,H}
4			

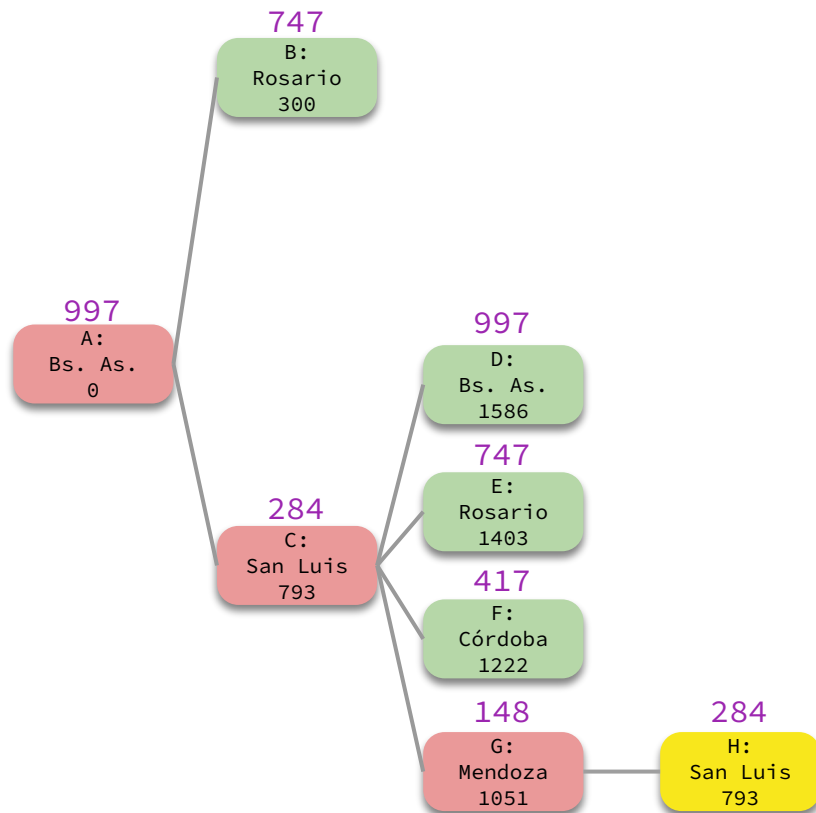




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G}
3	G	{B,D,E,F}	{B,D,E,F,H}
4	H	{B,D,E,F}	... CONTINÚA ...

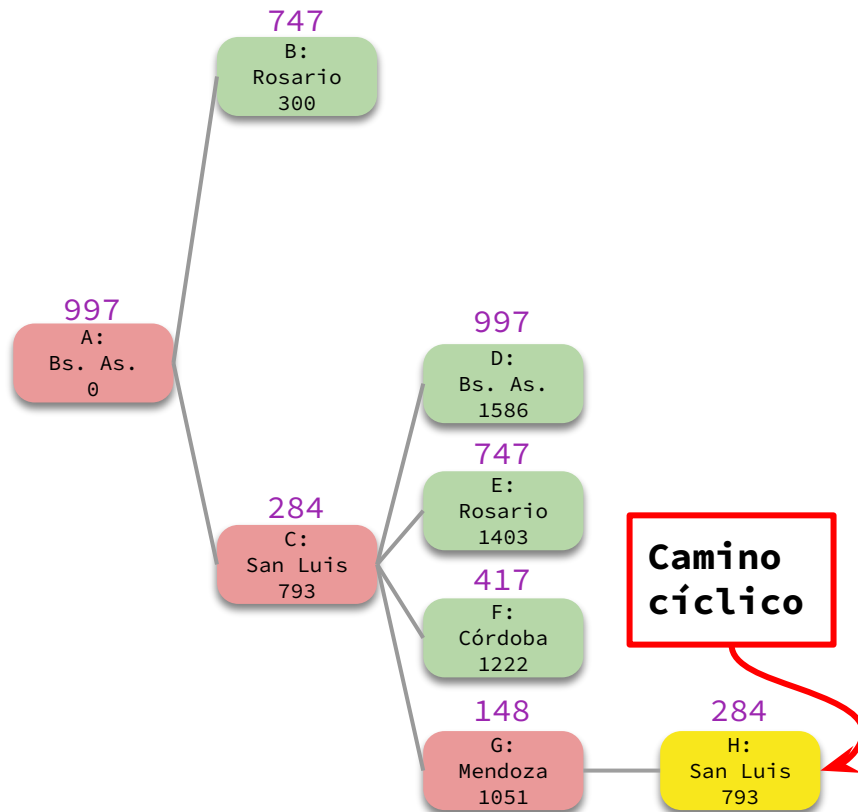




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F,G}
3	G	{B,D,E,F}	{B,D,E,F,H}
4	H	{B,D,E,F}	... CONTINÚA ...

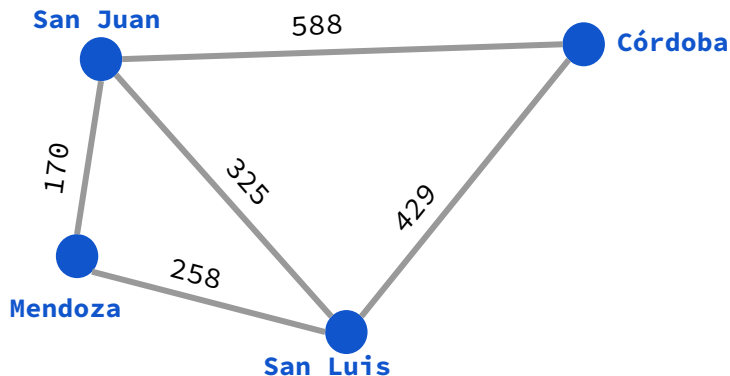




Ejemplo – Optimalidad

— — —

Encontrar el camino más corto de Córdoba a Mendoza en el siguiente mapa:



Valores heurísticos:

Córdoba = 498

San Luis = 232

San Juan = 143

Mendoza = 0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1			
2			
3			

498

A:
Córdoba.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{ }	
2			
3			

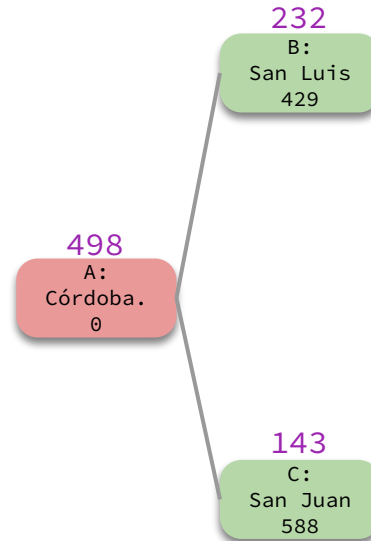
498
A:
Córdoba.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2			
3			

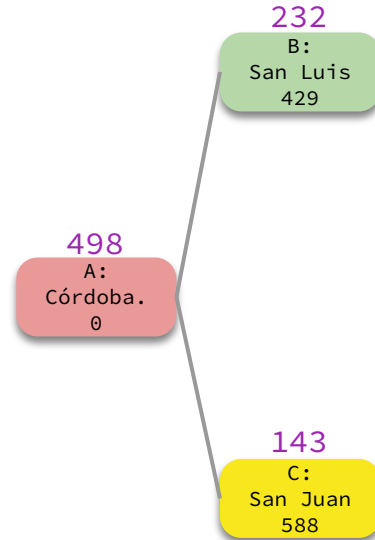




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	
3			

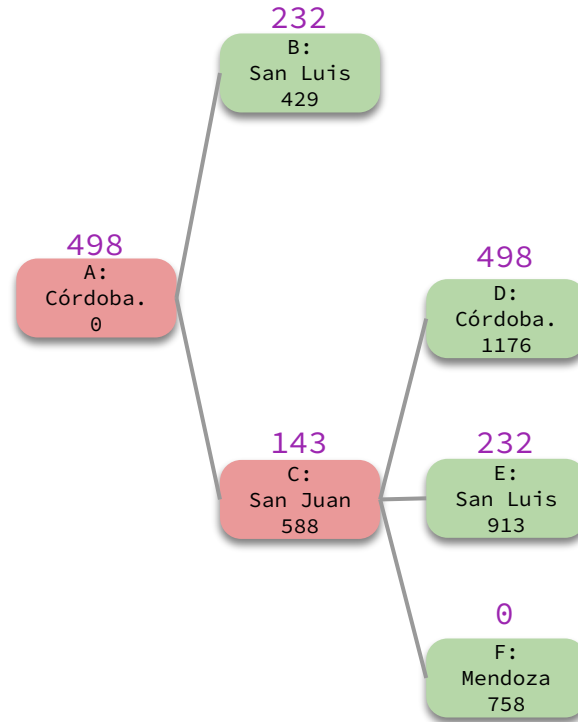




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F}
3			

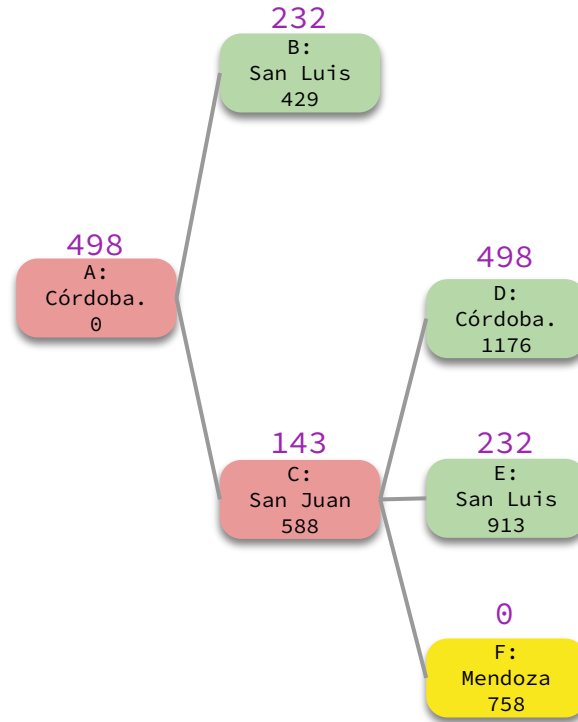




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F}
3	F	{B,D,E}	FIN



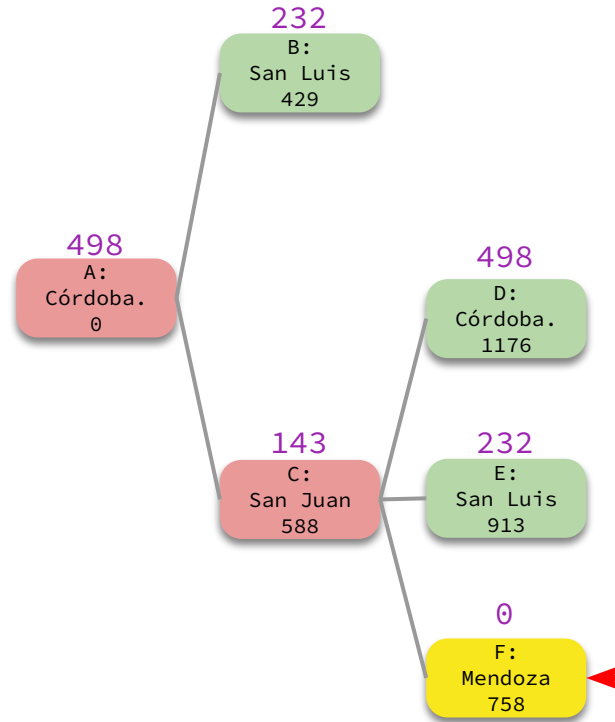


Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	C	{B}	{B,D,E,F}
3	F	{B,D,E}	FIN

Encontramos una solución subóptima. El camino Córdoba → San Luis → Mendoza tiene costo $687 < 758$



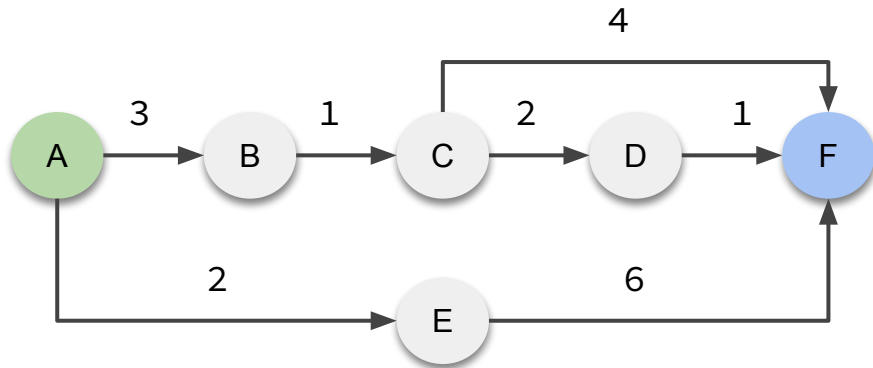


Ejercicio

Sea un problema descrito por el siguiente espacio de estados, donde A es el estado inicial y F es el estado objetivo.

Resolverlo con la estrategia GBFS, donde la heurística estima los siguientes costos para los estados:

A = 6, B = 3, C = 2, D = 1, E = 5, F = 0





Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1			
2			
3			
4			

6 A
0



Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	
2			
3			
4			

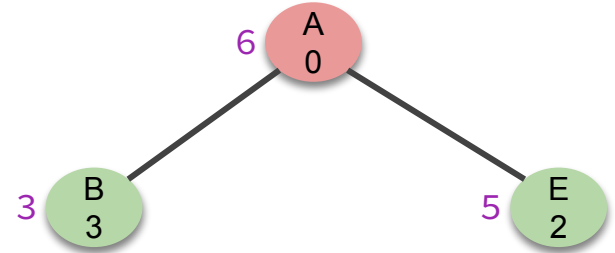
6 A
0



Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2			
3			
4			

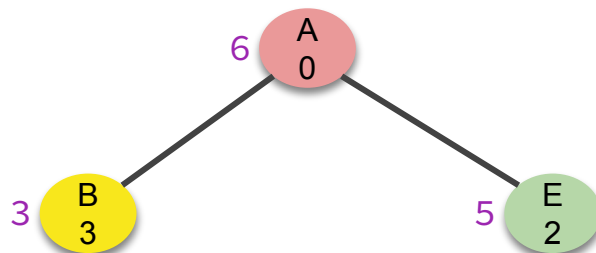




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	
3			
4			

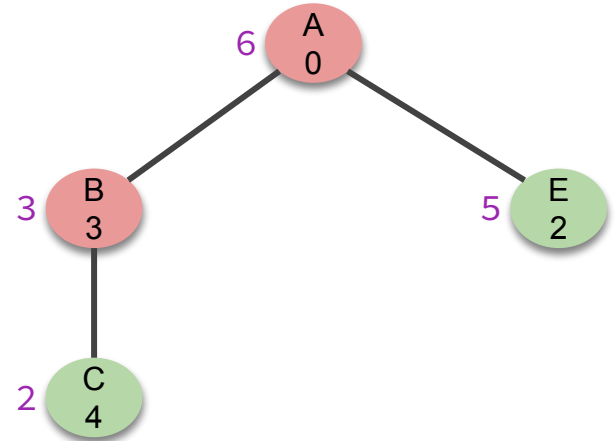




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3			
4			

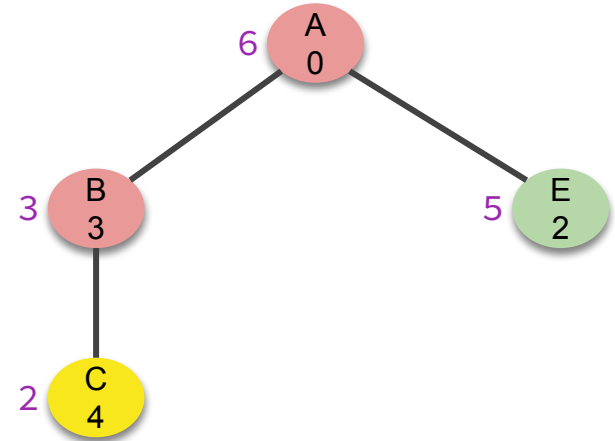




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	
4			

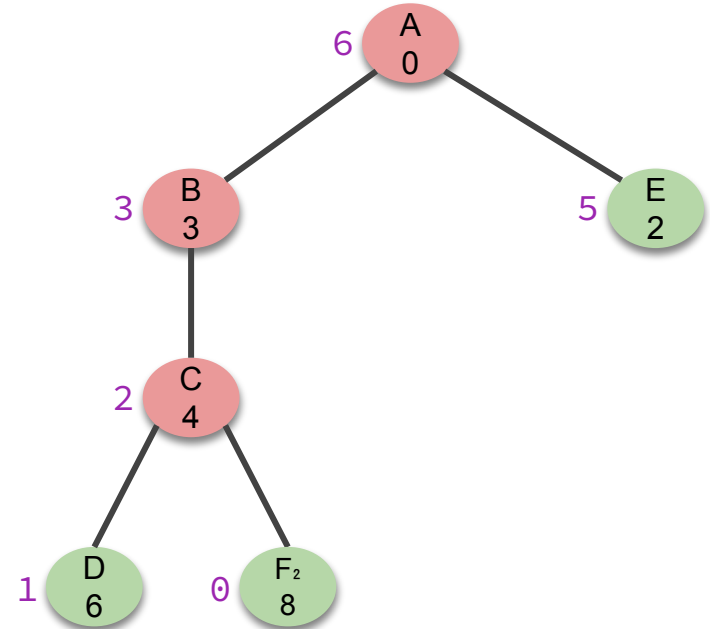




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4			

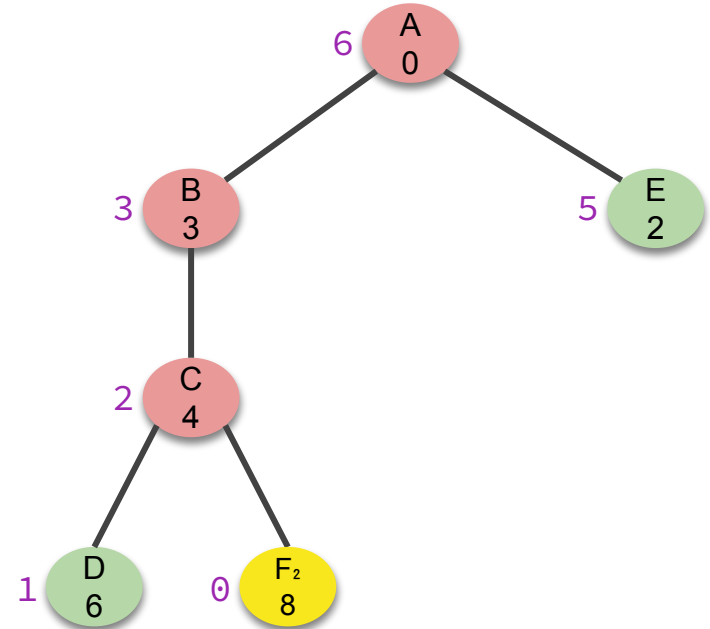




Respuesta

— — —

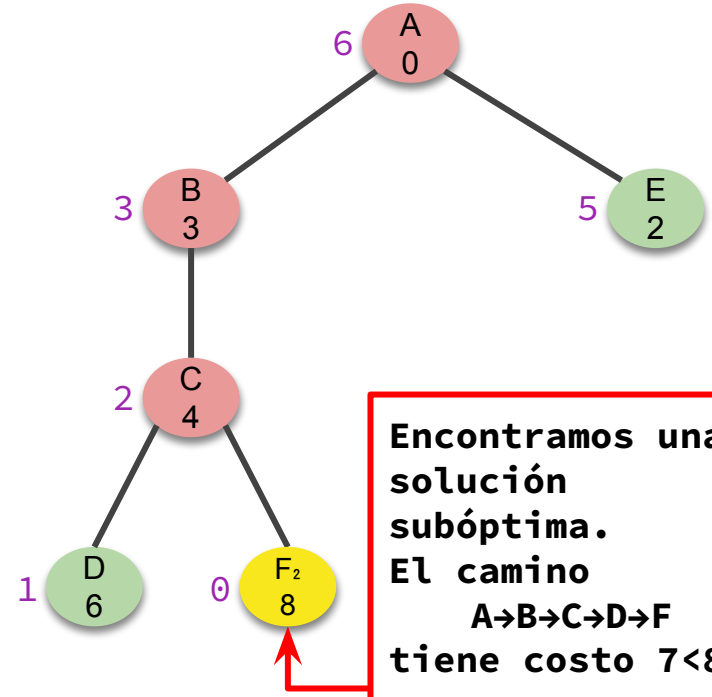
Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4	F ₂	{E,D}	FIN





Respuesta

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4	F ₂	{E,D}	FIN



Implementación

- ¿Cómo elegimos de la frontera el próximo nodo a expandir?

El nodo n con el menor costo $h(n)$.

- ¿Cómo lo logramos?

Al igual que en UCS, usando el TAD **cola de prioridad** para la frontera pero ordenando los nodos por la función heurística.



Algoritmo GBFS en grafos

— — —

```
1 function GRAPH-GBFS(problema, h) return solución o fallo
2   raíz ← Nodo(estado = problema.estado-inicial, costo = 0)
3   frontera ← ColaPrioridad()
4   frontera.encolar(raíz, h(raíz.estado))
5   alcanzados ← {raíz.estado: raíz}
6   do
7     if (frontera.vacia()) then return fallo
8     nodo ← frontera.desencolar()
9     if (problema.test-objetivo(nodo.estado)) then return solución(nodo)
10    forall acción in problema.acciones(nodo.estado) do
11      hijo ← Nodo(estado = problema.resultado(nodo.estado, acción),
12                  costo = nodo.costo + problema.c(nodo.estado, acción),
13                  padre = nodo, acción = acción)
14      if hijo.estado is not in alcanzados or hijo.costo < alcanzados[hijo.estado].costo then
15        alcanzados[hijo.estado] = hijo
16      frontera.encolar(hijo, h(hijo.estado))
```

Performance de TREE-GBFS y GRAPH-GBFS

— — —

- **Compleitud.**  (en árboles hay que detectar caminos cíclicos).
- **Optimalidad.** 
- **Tiempo y memoria.**
 - La cantidad de nodos generados puede reducirse drásticamente con una función heurística apropiada.

Entre UCS y GBFS

- UCS es óptima, pero genera todos los nodos con costo de camino menor al de una solución óptima.
- GBFS puede reducir drásticamente la cantidad de nodos generados con una función heurística apropiada, pero no es óptima.

Suena razonable combinarlas...

Búsqueda A*

A* Search

Combina las estrategias UCS y GBFS.

Dada una función heurística h , se expande siempre el nodo n de la frontera con la menor evaluación según una **función de evaluación** f definida por:

$$f(n) = n.\text{costo} + h(n),$$

es decir, $f(n)$ es el costo estimado de la solución menos costosa que pasa por n .

— — —



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1			
2			
3			
4			
5			

$0 + 997 = 997$

A:
Bs. As.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	
2			
3			
4			
5			

$$0 + 997 = 997$$

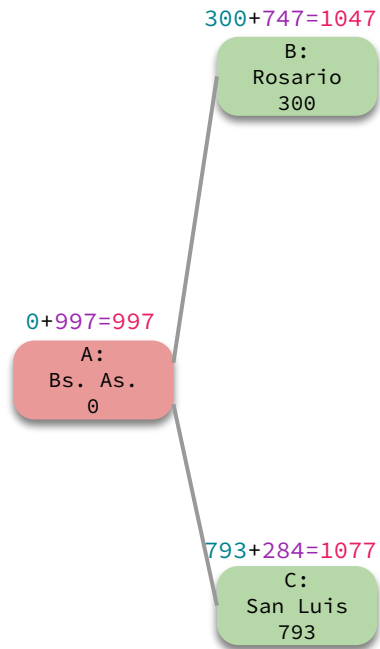
A:
Bs. As.
0



Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2			
3			
4			
5			

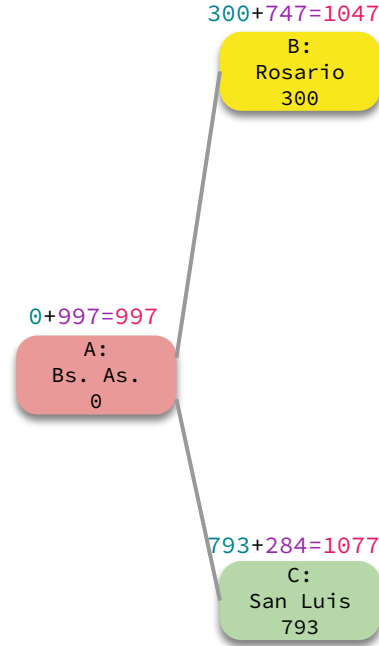




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	
3			
4			
5			

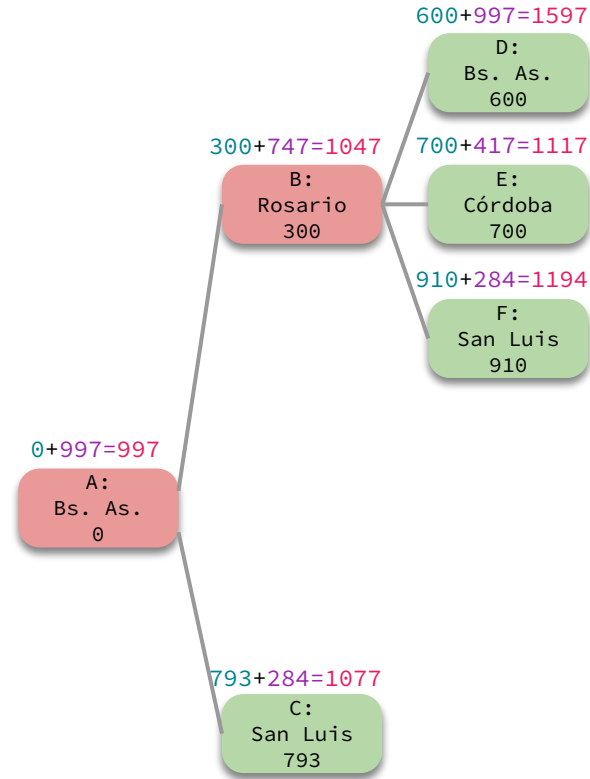




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3			
4			
5			

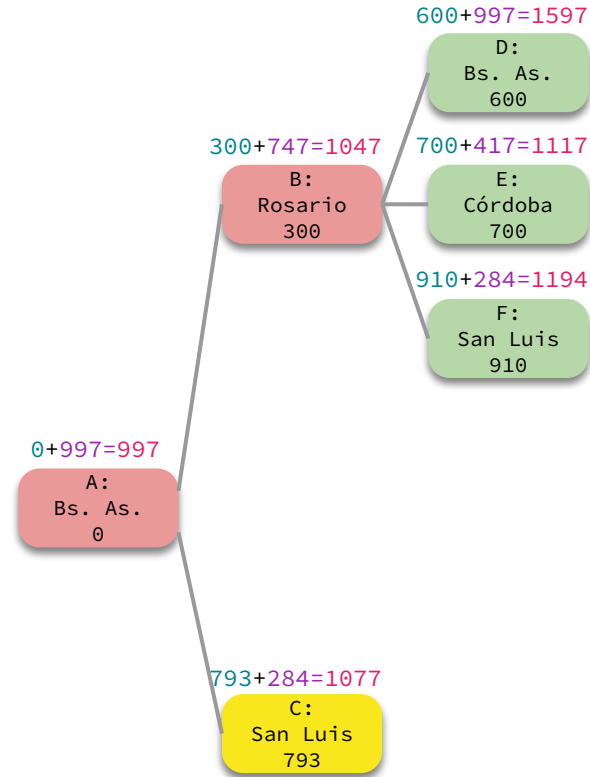




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3	C	{D,E,F}	
4			
5			

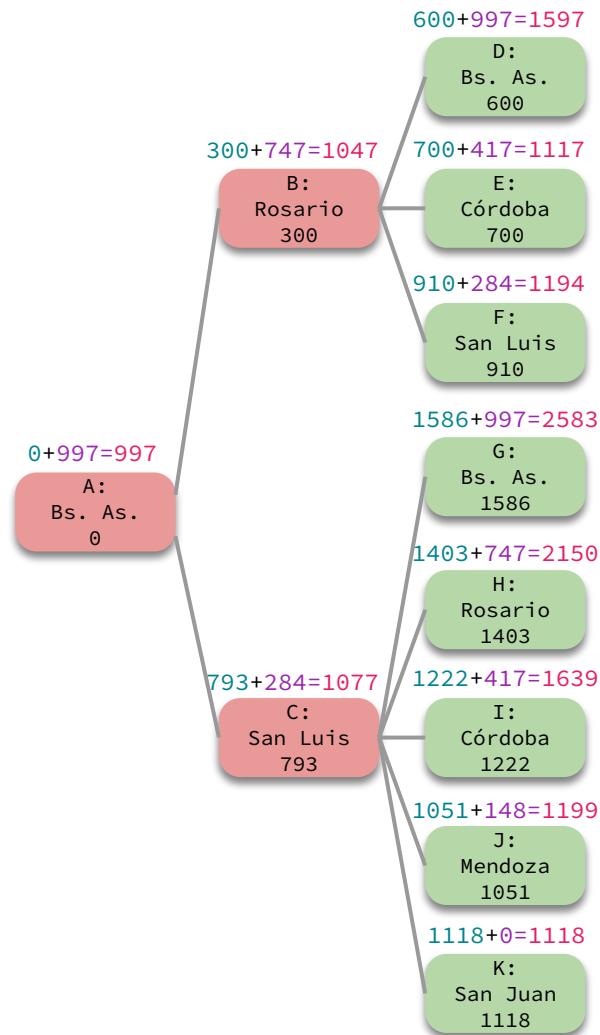




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3	C	{D,E,F}	{D,E,F,G,H,I,J,K}
4			
5			

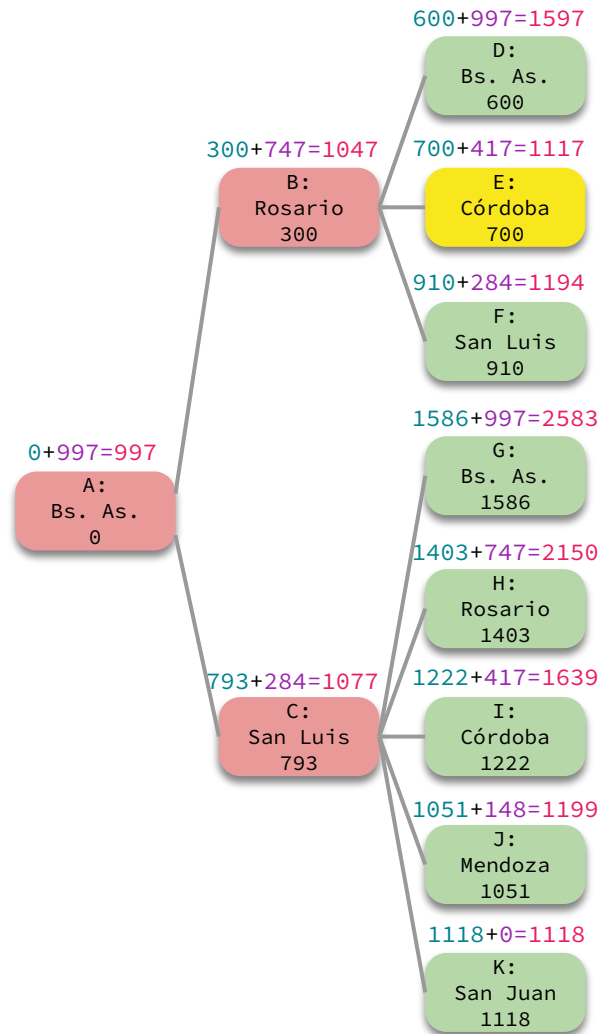




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3	C	{D,E,F}	{D,E,F,G,H,I,J,K}
4			
5			

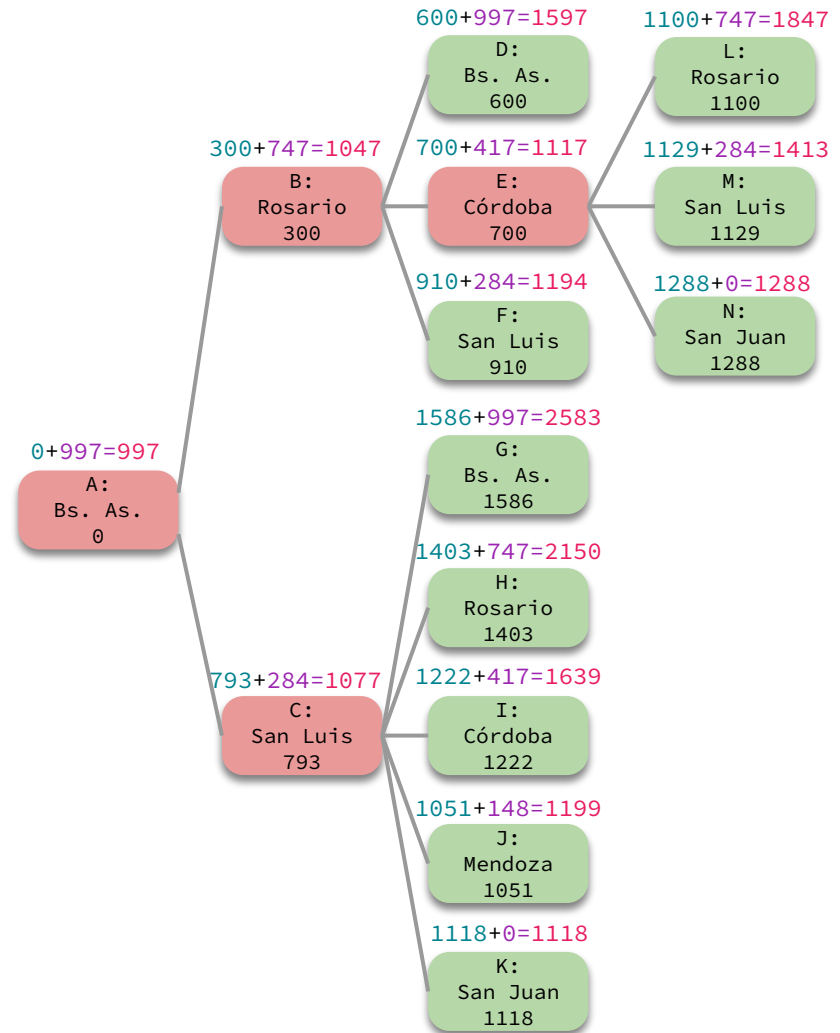




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3	C	{D,E,F}	{D,E,F,G,H,I,J,K}
4	E	{D,F,G,H,I,J,K}	{D,F,G,H,I,J,K,L,M,N}
5			

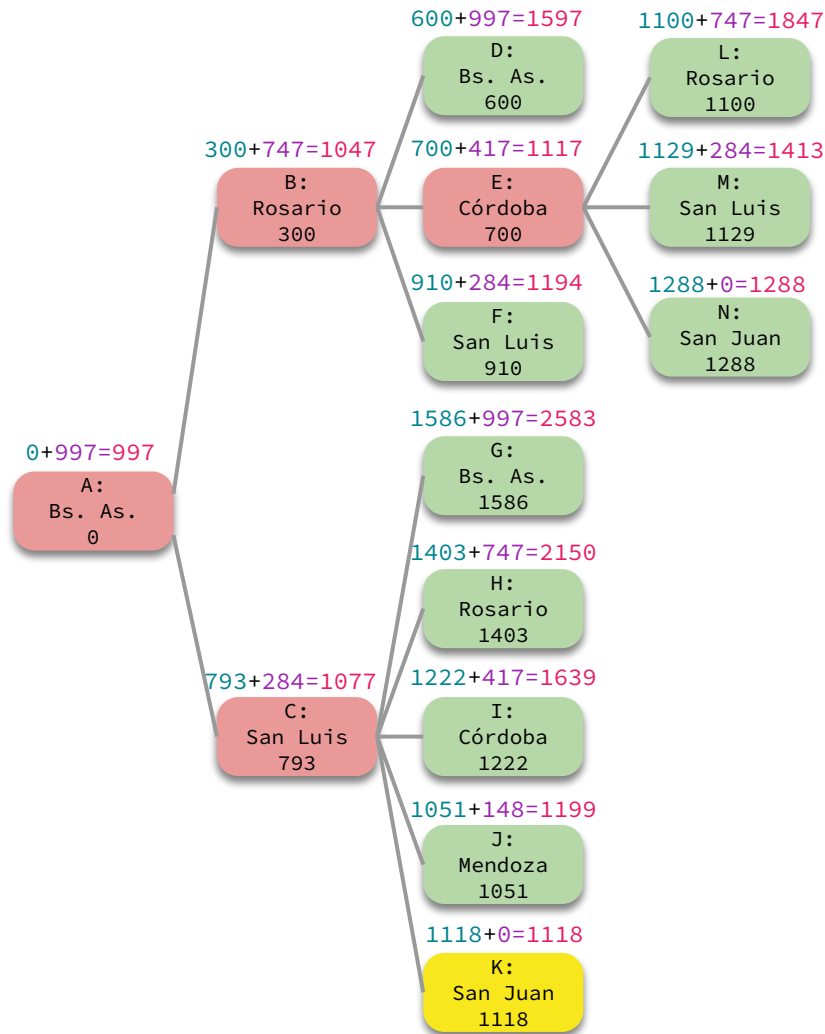




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3	C	{D,E,F}	{D,E,F,G,H,I,J,K}
4	E	{D,F,G,H,I,J,K}	{D,F,G,H,I,J,K,L,M,N}
5	K	{D,F,G,H,I,J,L,M,N}	FIN

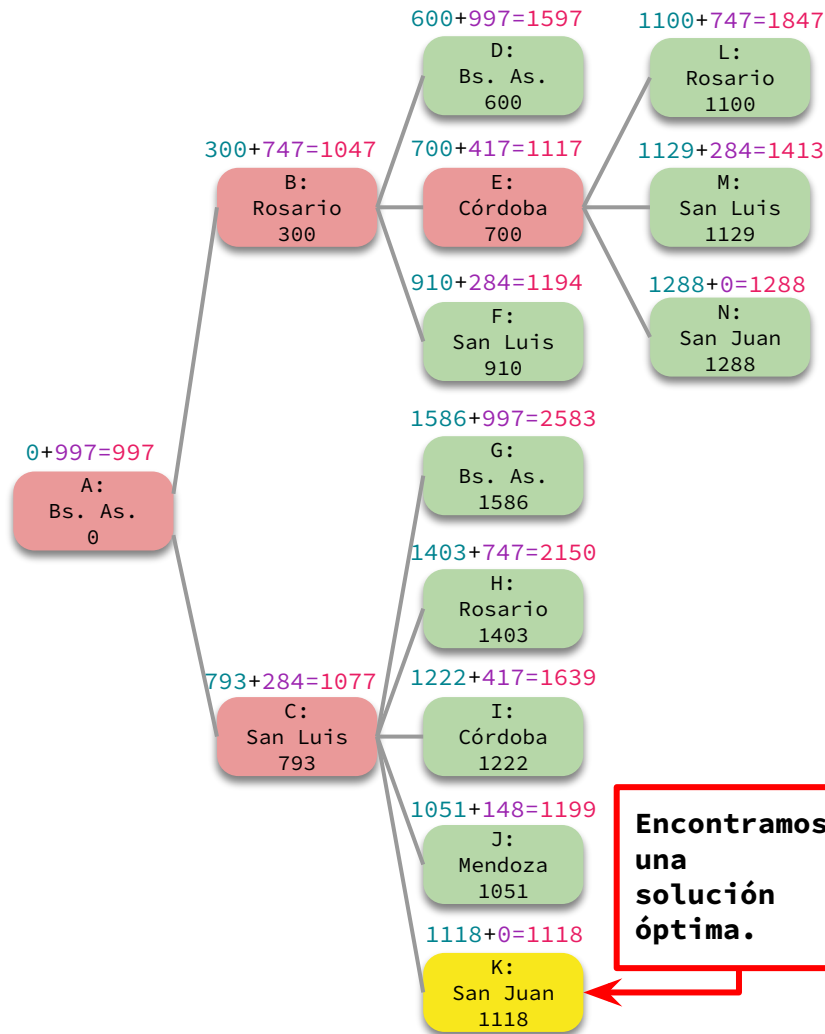




Ejemplo

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,C}
2	B	{C}	{C,D,E,F}
3	C	{D,E,F}	{D,E,F,G,H,I,J,K}
4	E	{D,F,G,H,I,J,K}	{D,F,G,H,I,J,K,L,M,N}
5	K	{D,F,G,H,I,J,L,M,N}	FIN



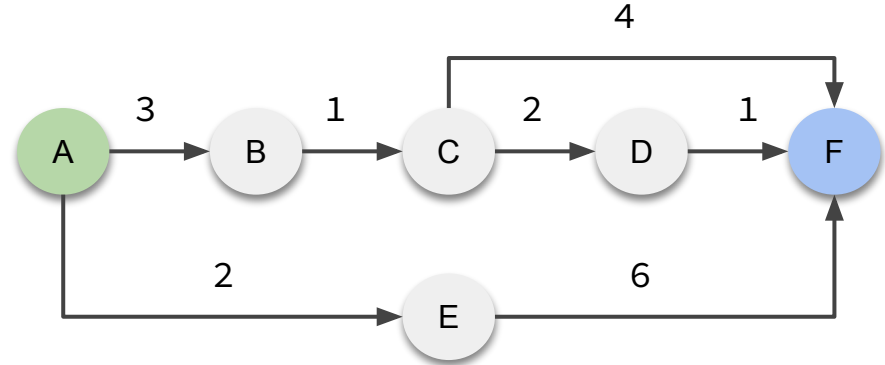


Ejercicio

Sea un problema descrito por el siguiente espacio de estados, donde A es el estado inicial y F es el estado objetivo.

Resolverlo con la estrategia A*, donde la heurística estima los siguientes costos para los estados:

A = 6, B = 3, C = 2, D = 1, E = 5, F = 0





Respuesta

— — —

$$0 + 6 = 6$$

A
0

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1			
2			
3			
4			
5			



Respuesta

— — —

$$0 + 6 = 6$$

A
0

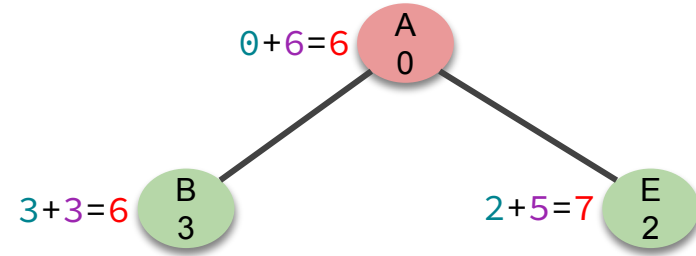
Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{ }	
2			
3			
4			
5			



Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2			
3			
4			
5			

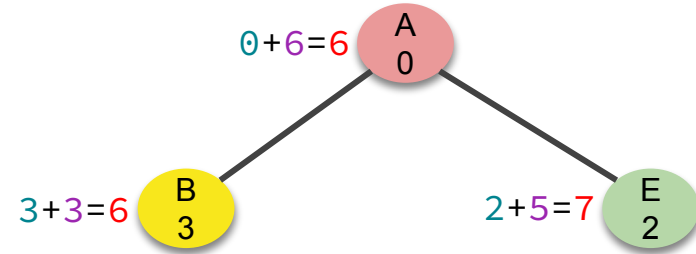




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	
3			
4			
5			

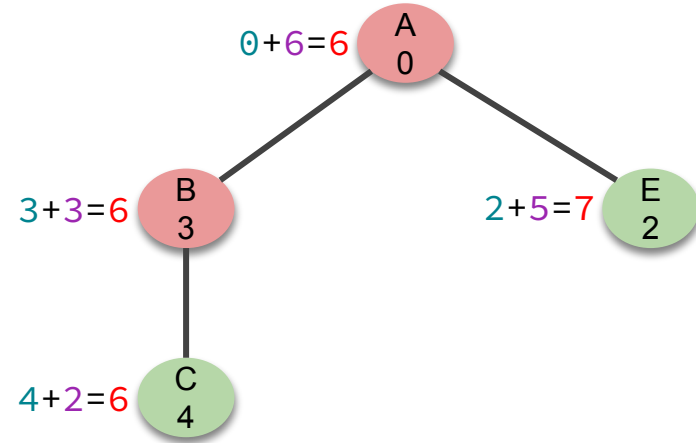




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3			
4			
5			

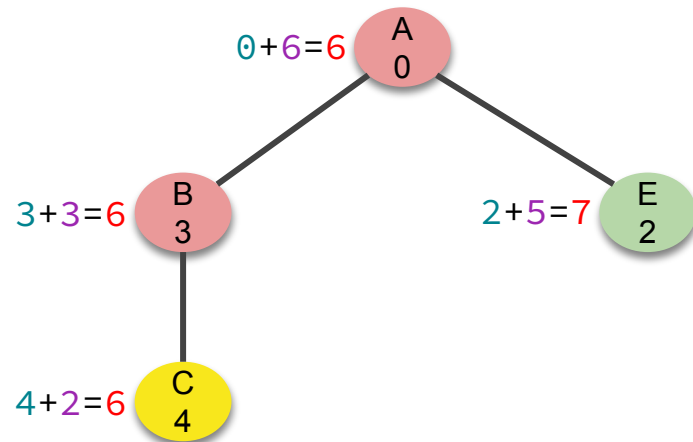




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	
4			
5			

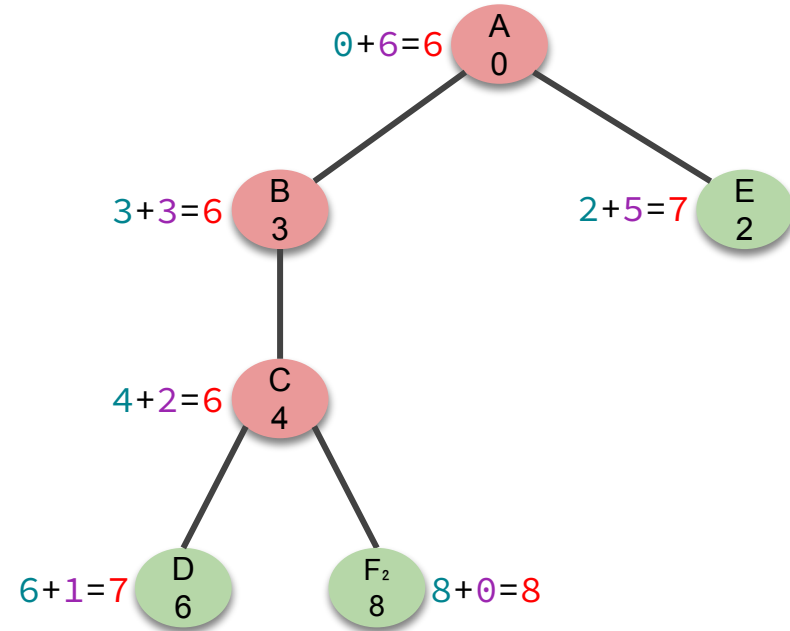




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4			
5			

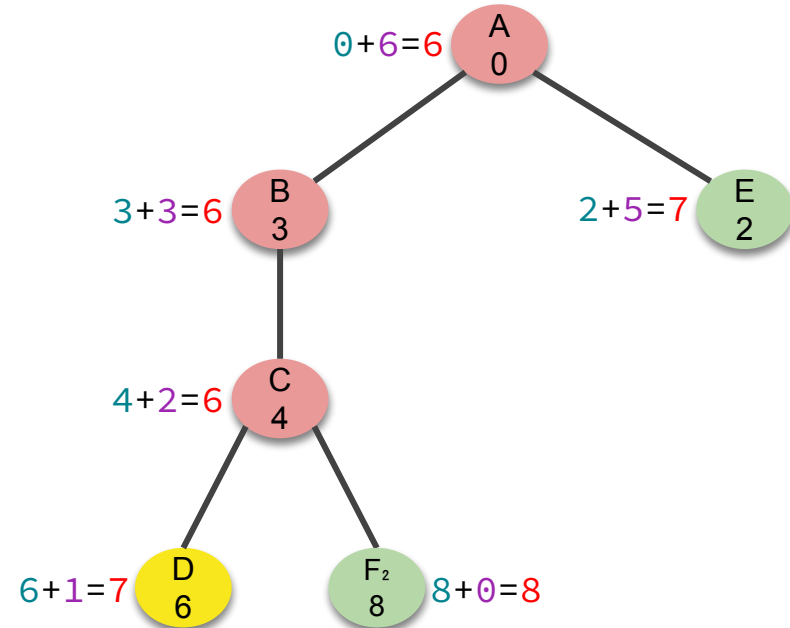




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4	D	{E,F ₂ }	
5			

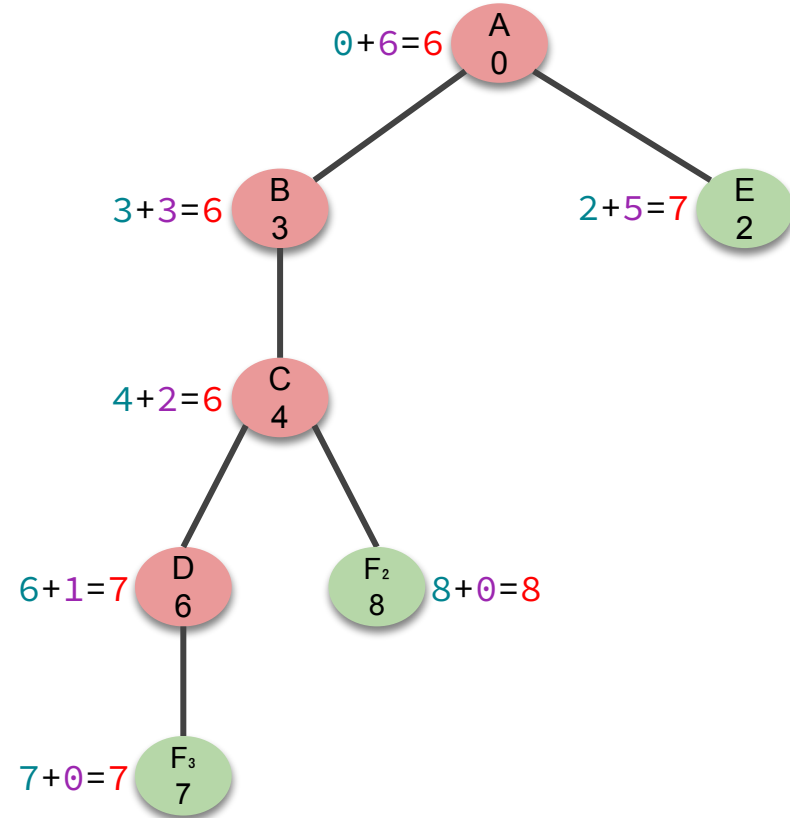




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4	D	{E,F ₂ }	{E,F ₂ ,F ₃ }
5			

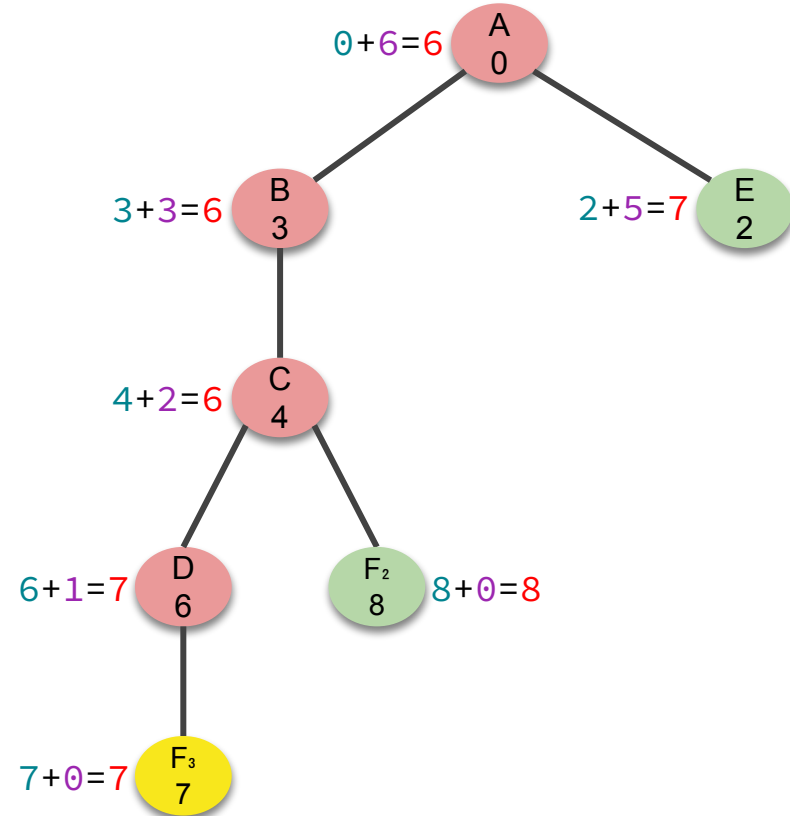




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4	D	{E,F ₂ }	{E,F ₂ ,F ₃ }
5	F ₃	{E,F ₂ }	FIN

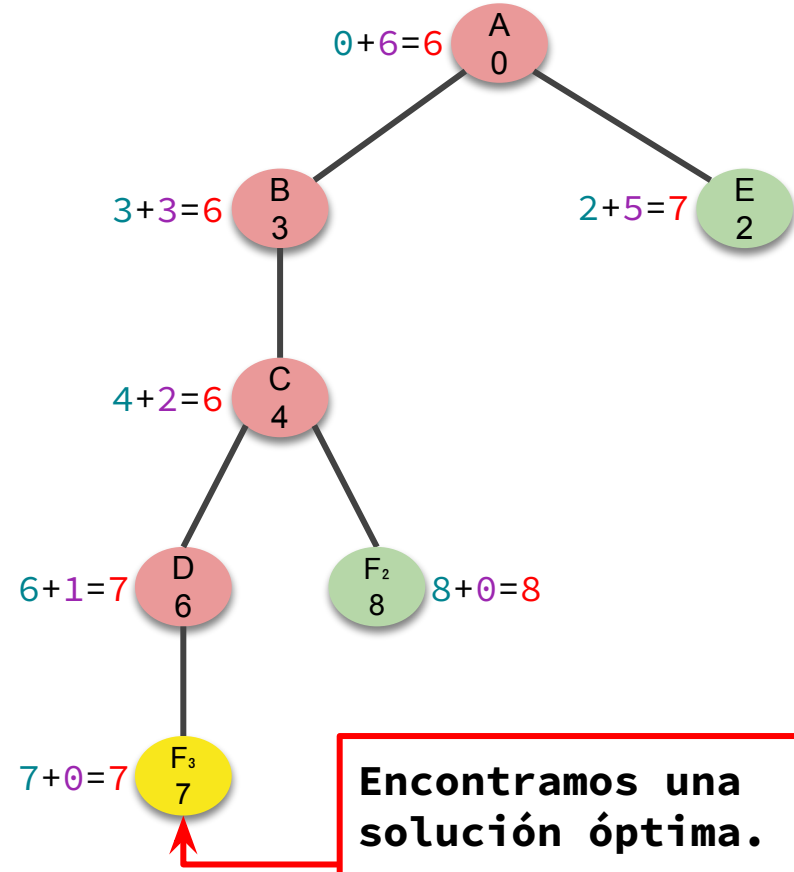




Respuesta

— — —

Nº	Nodo actual	Frontera antes de expandir	Frontera después de expandir
0	-	{A}	-
1	A	{}	{B,E}
2	B	{E}	{E,C}
3	C	{E}	{E,D,F ₂ }
4	D	{E,F ₂ }	{E,F ₂ ,F ₃ }
5	F ₃	{E,F ₂ }	FIN



Implementación

- ¿Cómo elegimos de la frontera el próximo nodo a expandir?

El nodo n con la menor evaluación $f(n)$.

- ¿Cómo lo logramos?

Al igual que en UCS y GBFS, usando el TAD **cola de prioridad** para la frontera pero ordenando los nodos por la función de evaluación f .

Algoritmo de búsqueda A* en grafos

— — —


```
1 function GRAPH-ASTAR(problema, h) return solución o fallo
2   raíz ← Nodo(estado = problema.estado-inicial, costo = 0)
3   frontera ← ColaPrioridad()
4   frontera.encolar(raíz, raíz.costo + h(raíz.estado))
5   alcanzados ← {raíz.estado: raíz}
6   do
7     if (frontera.vacía()) then return fallo
8     nodo ← frontera.desencolar()
9     if (problema.test-objetivo(nodo.estado)) then return solución(nodo)
10    forall acción in problema.acciones(nodo.estado) do
11      hijo ← Nodo(estado = problema.resultado(nodo.estado, acción),
12                  costo = nodo.costo + problema.c(nodo.estado, acción),
13                  padre = nodo, acción = acción)
14      if hijo.estado is not in alcanzados or hijo.costo < alcanzados[hijo.estado].costo then
15        alcanzados[hijo.estado] = hijo
16      frontera.encolar(hijo, hijo.costo + h(hijo.estado))
```

Admisibilidad y Consistencia

Una función heurística h es:

- **Admisible** si, para todo nodo n , $h(n)$ nunca sobrestima el costo de alcanzar un estado objetivo.
- **Consistente** si, para todo nodo n e hijo n' generado por una acción a , se cumple

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

-  **Ejemplo.** El costo de viaje en línea recta es una heurística admisible y consistente para el problema del camino más corto.
- Toda heurística consistente es admisible.

Performance de TREE-ASTAR y GRAPH-ASTAR

— — —

- **Complejidad.** ✓
- **Optimalidad.**
 - En árboles. ✓ si la heurística es admisible.
 - En grafos. ✓ si la heurística es consistente.



Resumen

- ❑ Vimos dos estrategias de búsqueda informadas: GBFS y A*.
- ❑ GBFS expande siempre el nodo no expandido con menor costo heurístico $h(n)$. No es óptima pero a menudo es bastante eficiente.
- ❑ A* expande siempre el nodo no expandido con menor costo de evaluación $f(n) = n.\text{costo} + h(n)$. Es óptima si la heurística es admisible (en árboles) y consistente (en grafos).



Próximamente

— — —

¿Cómo generar buenas heurísticas para un problema?

Juego del 8 (8 Puzzle)

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Descripción

— — —

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Objetivo. Alcanzar una configuración objetivo.

Reglas. Una ficha se puede mover de una casilla A a una casilla B si:

1. A está adyacente (verticalmente u horizontalmente) a B , y
2. B es una casilla vacía.



Ejemplo

— — —

7	2	4
5		6
8	3	1

6 izq.



7	2	4
5	6	
8	3	1

1 arr.



7	2	4
5	6	1
8	3	

3 der.



7	2	4
5	6	1
8		3

Formulación

— — —

- **Estados.** Matrices 3x3 con números del 1 al 8 y una casilla vacía. Hay $9! = 362.880$ estados, de los cuales solo la mitad son alcanzables.
- **Estado inicial.** Cualquier estado puede ser designado como inicial.
- **Acciones.** \leftarrow , \uparrow , \rightarrow , \downarrow (a dónde se mueve el espacio vacío).
- **Modelo transicional.** Por ejemplo, \leftarrow intercambia en la matriz la casilla vacía con el elemento de la casilla de su izquierda.
- **Test objetivo.** Verificar si el estado coincide con la configuración deseada.
- **Costo de camino.** El costo individual es 1, luego el costo de camino es el número de pasos realizados.

Resolución mediante búsqueda

Dado un estado inicial generado aleatoriamente.

- Menor profundidad de un nodo objetivo: **22 pasos** (en promedio).
- Factor de ramificación: **3** (en promedio).
- El árbol de búsqueda hasta el nivel 22 tiene aproximadamente
 $3^{23} - 1 \approx 94.143.178.826 \approx 9,41 \times 10^{10}$ **nodos**
- Es conveniente cambiar a una búsqueda en grafos, ya que el espacio de estados tiene sólo **181.440** estados.
- El juego del 15 (15 puzzle) tiene **10^{13}** estados.
- **Es necesario pasar a una búsqueda informada, para ello es necesario definir una buena heurística.**

Heurísticas

Para un nodo n con el estado:

7	2	4
5		6
8	3	1

1. h_1 = número de fichas mal ubicadas.

📖 **Ejemplo.** $h_1(n) = 8$ (todas están mal ubicadas).

2. h_2 = suma de las distancias en vertical y horizontal de las fichas a su posición objetivo. **Distancia de Manhattan.**

📖 **Ejemplo.** $h_2(n) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

	↑	↑	↑	↑	↑	↑	↑	
Ficha:	1	2	3	4	5	6	7	8



Ambas son **admisibles**.



Ejercicio

— — —

Dado un nodo n con el siguiente estado:

	1	3
2	4	6
7	8	5

calcular $h_1(n)$ y $h_2(n)$.



Respuesta

	1	3
2	4	6
7	8	5

$h_1(n) = 6$ (solamente las fichas 1 y 4 están bien ubicadas).

$$h_2(n) = 0 + 3 + 3 + 0 + 1 + 3 + 1 + 1 = 12$$

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

Ficha: 1 2 3 4 5 6 7 8

h_1 vs. h_2

— — —

d	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	3644035	227	73
14	—	539	113
16	—	1301	211
18	—	3056	363
20	—	7276	676
22	—	18094	1219
24	—	39135	1641

Cada fila de la tabla reporta el número promedio de nodos generados por cada estrategia para 100 instancias aleatorias con solución de costo d .

h_1 vs. h_2

— — —

- ¿ h_2 es siempre mejor que h_1 ? **Si**.
- Para todo nodo n , $h_2(n) \geq h_1(n)$, luego h_2 **domina** a h_1 .
- A* con h_2 nunca expande más nodos que A* con h_1 .

¿Cómo generar heurísticas de este tipo?

Problemas relajados

Un **problema relajado** se
obtiene quitando
restricciones a las
acciones de un problema.

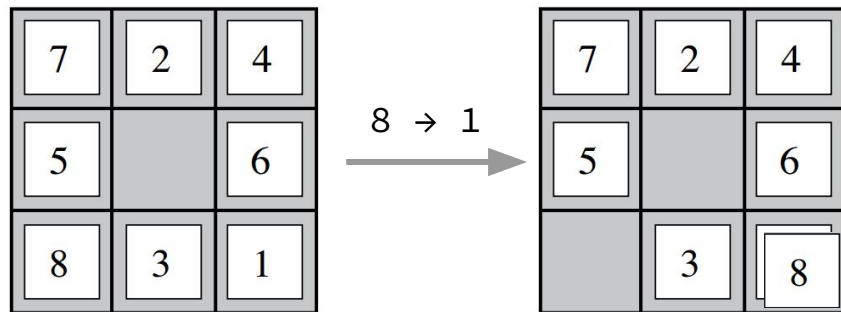
— — —



Ejemplo

Una ficha se puede mover de una casilla A a una casilla B si:

1. ~~A está adyacente (verticalmente u horizontalmente) a B , y~~
2. ~~B es una casilla vacía.~~



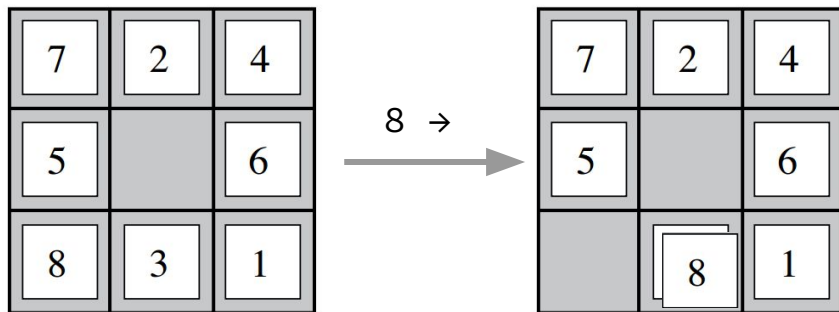
h_1 = número de fichas mal ubicadas. En este problema relajado, $h_1(n)$ da el **costo exacto** del camino de menor costo desde n al nodo objetivo.



Ejemplo

Una ficha se puede mover de una casilla A a una casilla B si:

1. A está adyacente (verticalmente u horizontalmente) a B , y
2. ~~B es una casilla vacía.~~



h_2 = distancia de Manhattan. En este problema relajado, $h_2(n)$ da el **costo exacto** del camino de menor costo desde n al nodo objetivo.

Admisibilidad y Consistencia

— — —

Sean P un problema con una heurística h y P' un problema relajado de P .

- El grafo de espacio de estados de P' tiene más arcos que el de P .
- Una solución óptima de P es una solución de P' , pero P' puede tener mejores soluciones si los arcos adicionales proveen caminos más cortos.
- Si $h(n)$ da el costo exacto del camino más corto desde n a un nodo objetivo en P' , entonces h es **admisible** y **consistente** para P .
- **Es crucial que P' pueda resolverse sin búsqueda. De lo contrario, los valores de h serían costosos de obtener.**

Subproblema

Un **subproblema** consiste en resolver una parte de un problema más grande.

En general, se obtiene quitando restricciones del objetivo de un problema.

— — —



Ejemplo

— — —

Ubicar las fichas 1, 2, 3, 4 y la casilla vacía en su posición correcta.

*	2	4
*		*
*	3	1

	1	2
3	4	*
*	*	*

La elección de 1-2-3-4 es arbitraria. Podría ser 5-6-7-8, 2-4-6-8, etc.

Modelo de bases de datos – Pattern database

— — —

Sea un subproblema P' de un problema P .

- El costo de una solución óptima de P' es una cota inferior del costo de una solución óptima de P .
- Un **modelo de bases de datos** almacena el costo de una solución óptima de P' para cualquier estado inicial.
- Dado un nodo n , se define una heurística h para P tal que $h(n)$ es el costo almacenado en la base de datos para la configuración del subproblema correspondiente.
- h es **admisible** para P .
- **Es crucial que P' pueda resolverse fácilmente mediante búsqueda, ya que debemos resolverlo para toda configuración inicial.**

Modelo de bases de datos disjuntas

— — —

- ¿Las heurísticas obtenidas de las bases de datos 1-2-3-4 y 5-6-7-8 se pueden sumar? **No es tan sencillo**, las soluciones a estos subproblemas seguro comparten movimientos.
- Para poder sumarlos, en el subproblema 1-2-3-4 no debemos registrar el costo total, sino solamente el número de movimientos que involucran las fichas 1-2-3-4 (y análogamente con el subproblema 5-6-7-8).
- Los **modelos de bases de datos disjuntas** utilizan esta idea.
- Permiten resolver el 15-puzzle en milisegundos.

Heurística compuesta

Sean h_1, \dots, h_m heurísticas para un problema y tales que ninguna domina la otra.

Se define la **heurística compuesta**

$$h(n) = \max\{h_1(n), \dots, h_m(n)\}.$$

— — —

Admisibilidad y Consistencia

— — —

- Si h_1, \dots, h_m son admisibles, entonces h es **admisibile**.
- Si h_1, \dots, h_m son consistentes, entonces h es **consistente**.
- h **domina** a h_1, \dots, h_m .



Resumen

— — —

- ❏ Vimos 3 formas de construir buenas heurísticas.
 - a. Relajando el problema.
 - b. Almacenando en una base de datos los costos de soluciones precomputadas para un subproblema.
 - c. Componiendo heurísticas disponibles.



Próximamente

— — —

Más allá de la búsqueda clásica...

Veremos algoritmos de **búsqueda local**: evalúan y modifican uno o varios estados actuales, en lugar de explorar sistemáticamente caminos desde un estado inicial