Inteligencia Artificial

Espacio de estados y búsqueda

Basado en:

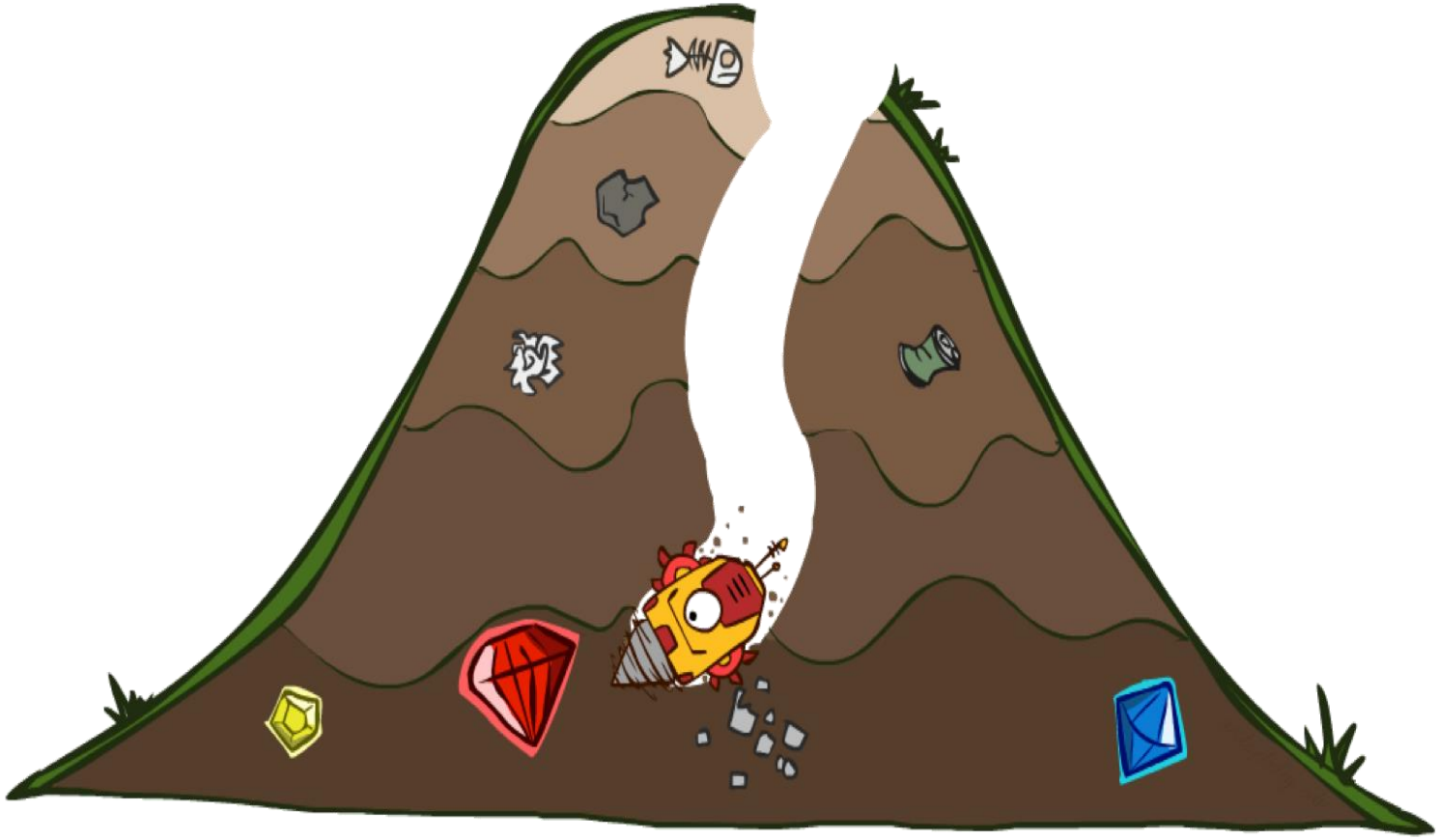
- Introduction to IA CS188, Univ. Berkeley

ROBOTIKA & SISTEMA AUTONOMOEN IKERKETA TALDEA

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Índice

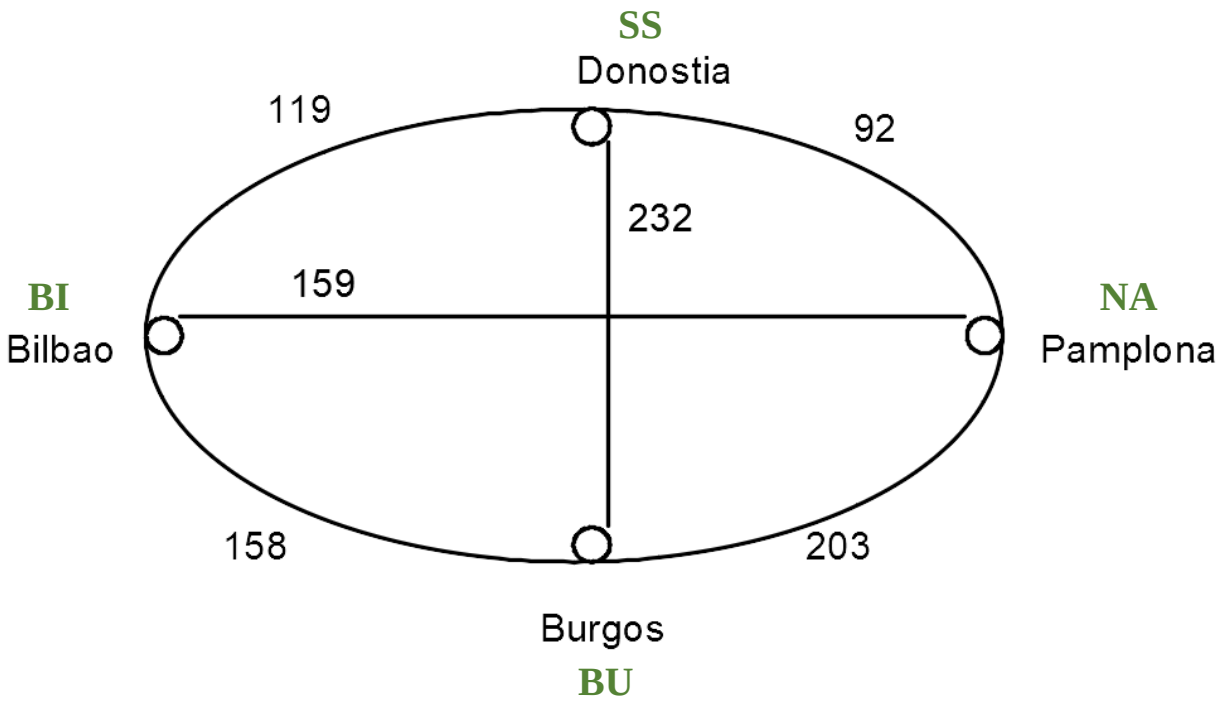
Búsqueda A* (A* Search)



Ramificar y acotar

- Hasta el momento, una vez tomada la decisión ya no hay vuelta atrás! Existe alguna forma de retroceder?

- Problema ejemplo - agente viajero

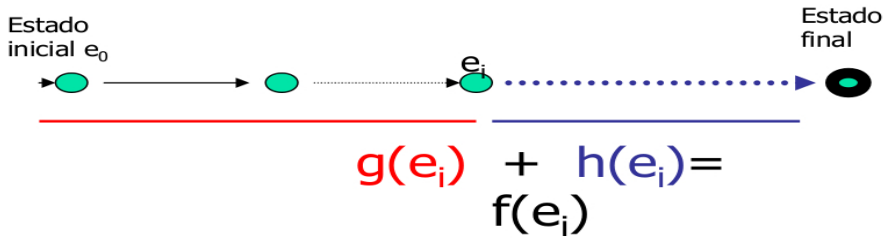


- Ramificar y acotar *(branch and bound)*
  - Cuatro versiones

Búsqueda guiada por información

- Heurístico ***h(e)***: estimación del coste de lo que falta para llegar desde el estado ***e*** hasta el nodo solución.
- El término ***h*(e)*** se define como el **coste real** que existe entre un nodo ***e*** y un nodo **solución** (el más barato).
- Coste ***g(e)***: coste real del camino que lleva del nodo inicial al estado ***e***
- Siendo ***e*** el estado final del problema:
  - **f(e)=g(e) + h(e)**, representa el ***coste estimado*** del camino que lleva hasta dicha solución
  - **f*(e)=g(e)+h*(e)** es el ***coste real*** del camino a la solución, desde el estado inicial hasta el estado final

- **Se plantea:**
  - Encontrar una configuración de éxito sobre el espacio de estados que minimice una función de evaluación de costes ***f = g + h***



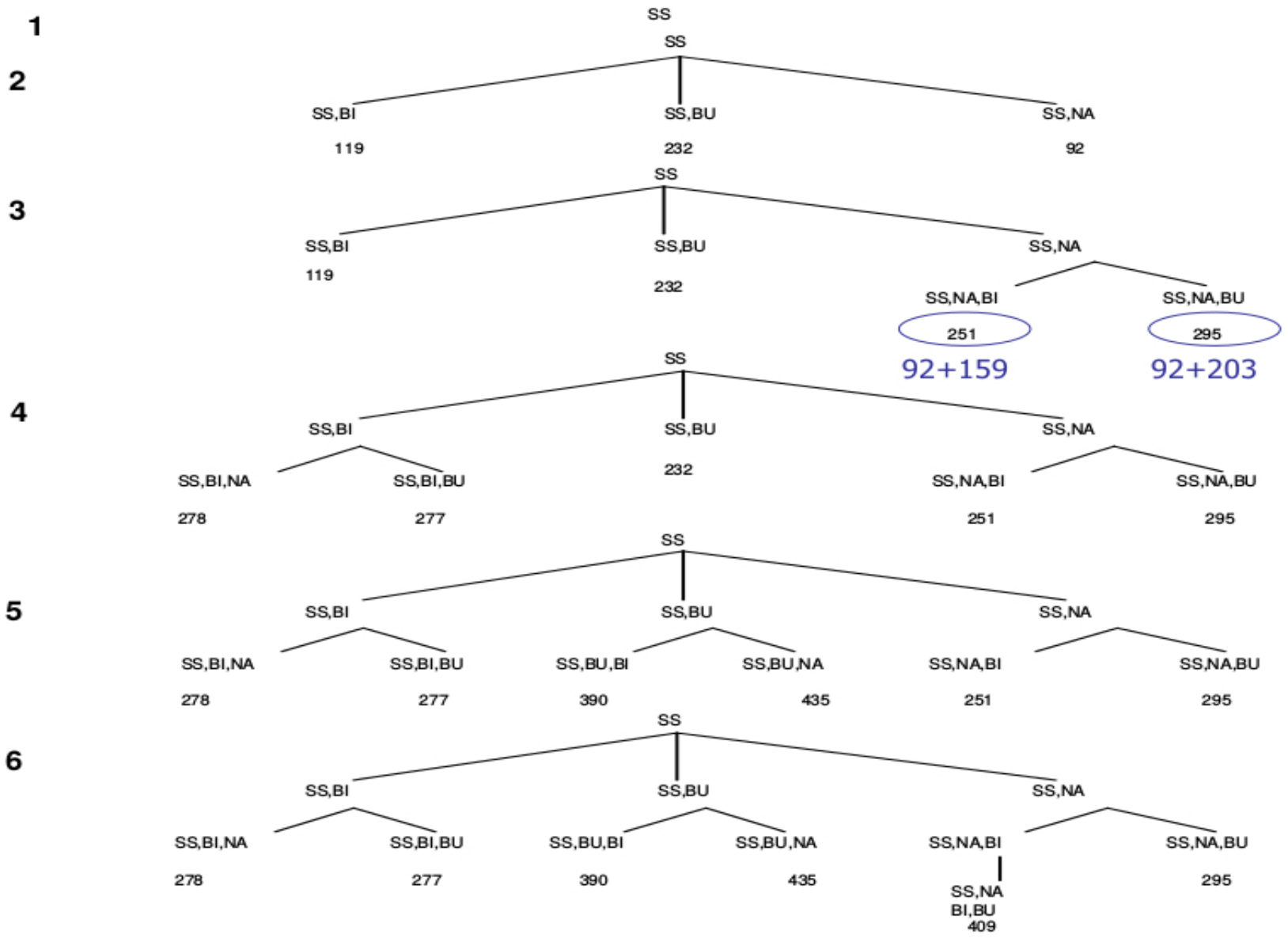
Ramificar y acotar v1

➢ Algoritmo

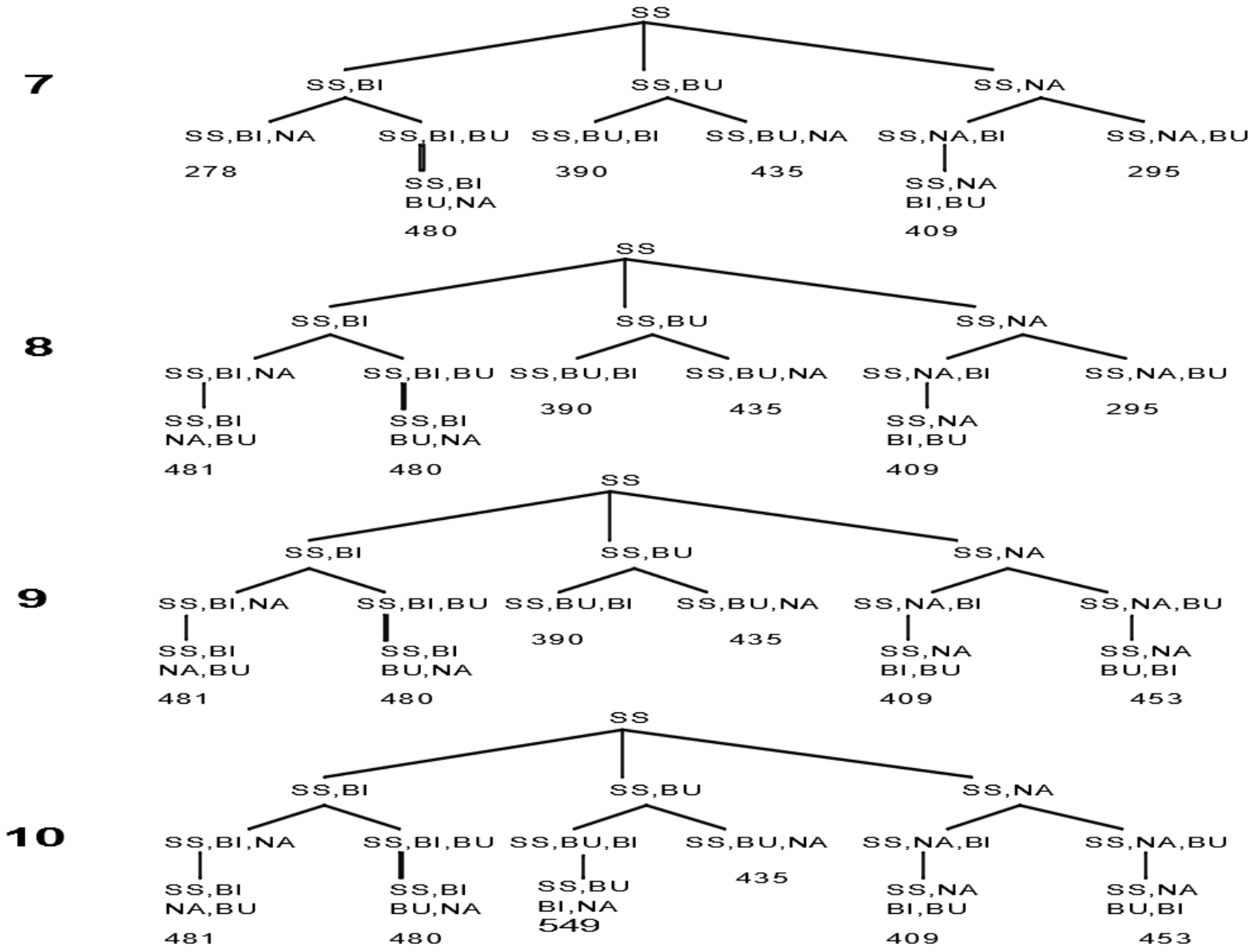
- Construir una lista de caminos parciales con el nodo raíz (**coste acumulado = g = 0**).
- Hasta que la lista esté vacía o (el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino)
  - Eliminar primer camino de la lista
  - Formar nuevos caminos a partir del eliminado añadiendo tanto caminos como hijos tenga el último nodo de ese camino
  - Añadir esos nuevos caminos a la lista junto **con su coste**
  - Ordenar la lista de menor a mayor considerando la función del coste en ese momento

➢ Si el primer camino de la lista encuentra el nodo objetivo, anunciar éxito, sino, fallo.

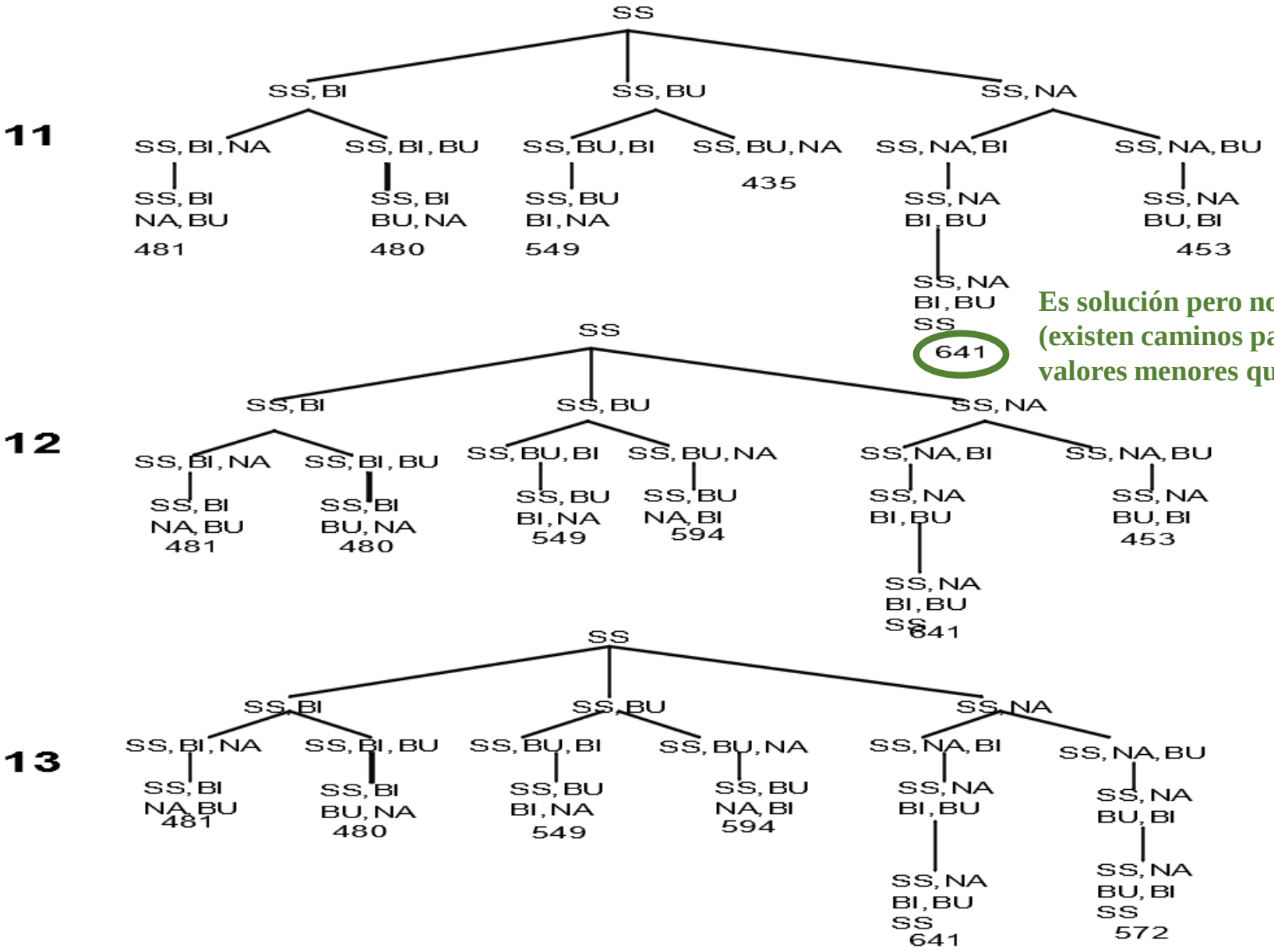
Ramificar y acotar v1



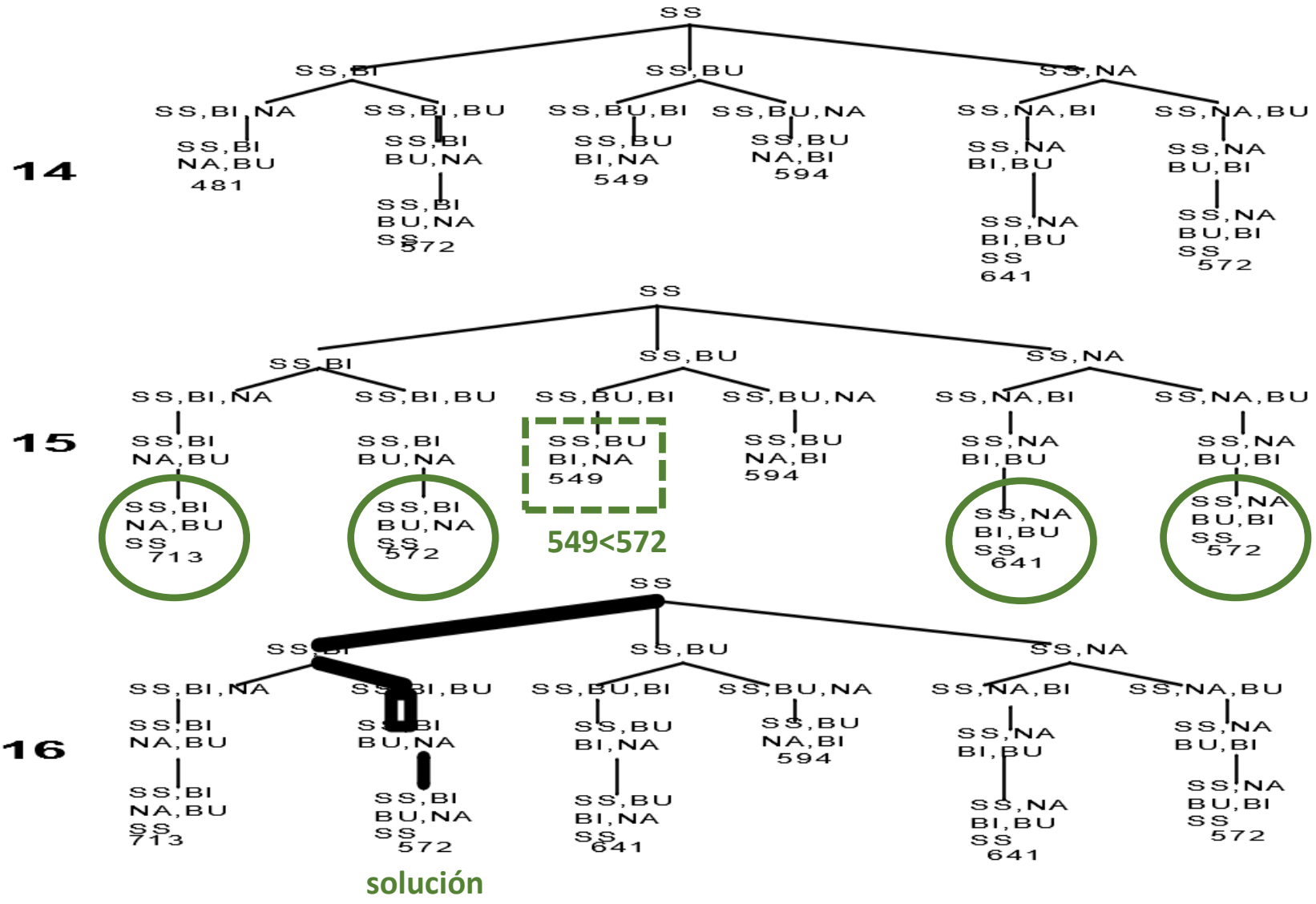
Ramificar y acotar v1



Ramificar y acotar v1



Ramificar y acotar v1



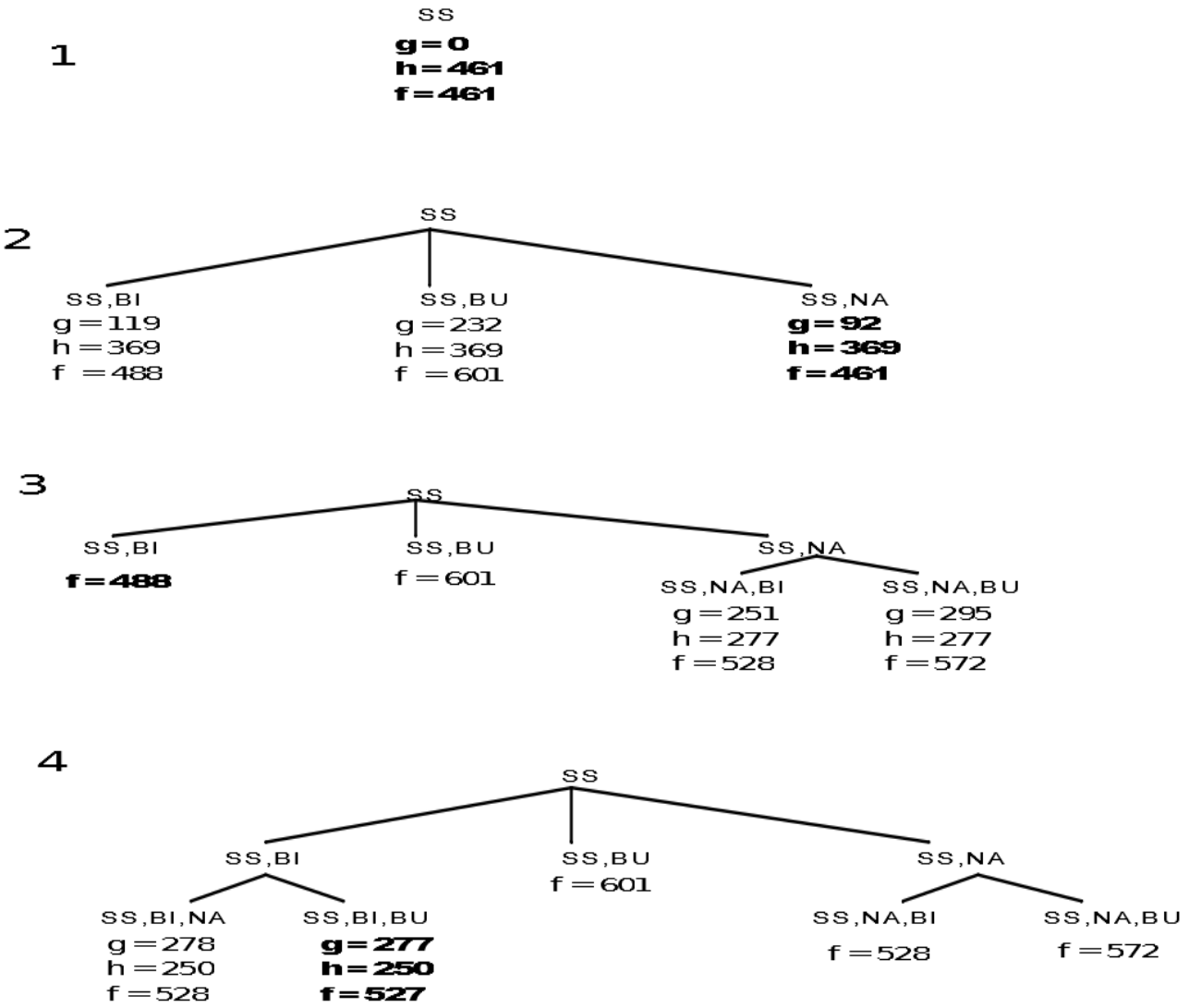
Ramificar y acotar v2

➢ Algoritmo

- Construir una lista de caminos parciales con el nodo raíz (***f = g + h*** ; ***g***=coste acumulado, ***h***=coste heurístico hasta la solución).
- Hasta que la lista esté vacía o (el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino)
  - Eliminar primer camino de la lista
  - Formar nuevos caminos a partir del eliminado añadiendo tanto caminos como hijos tenga el último nodo de ese camino
  - Añadir esos nuevos caminos a la lista junto con su **coste total (*f*)**
  - Ordenar la lista de menor a mayor considerando la función del coste en ese momento

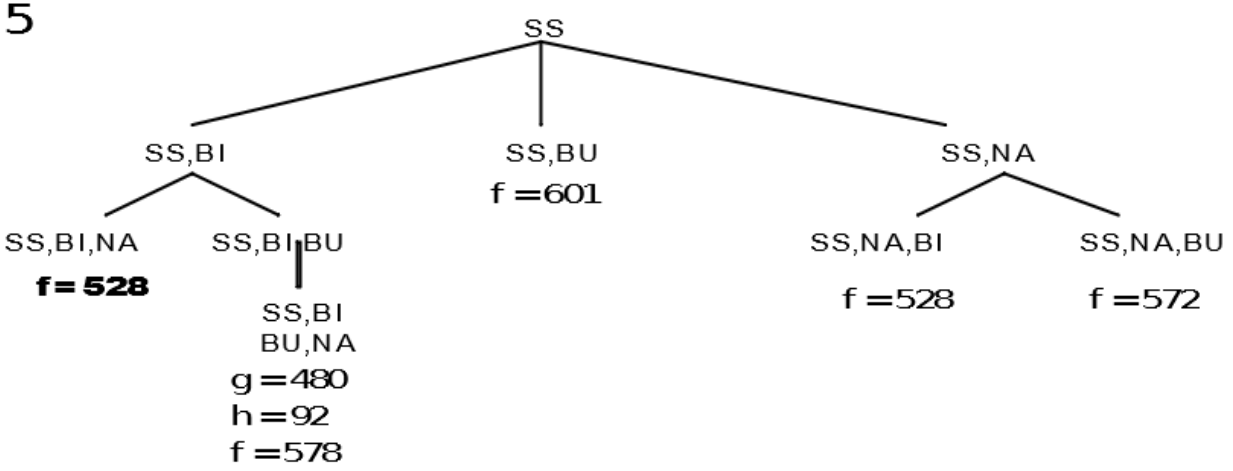
➢ Si el primer camino de la lista encuentra el nodo objetivo, anunciar éxito, sino, fallo.

Ramificar y acotar v2

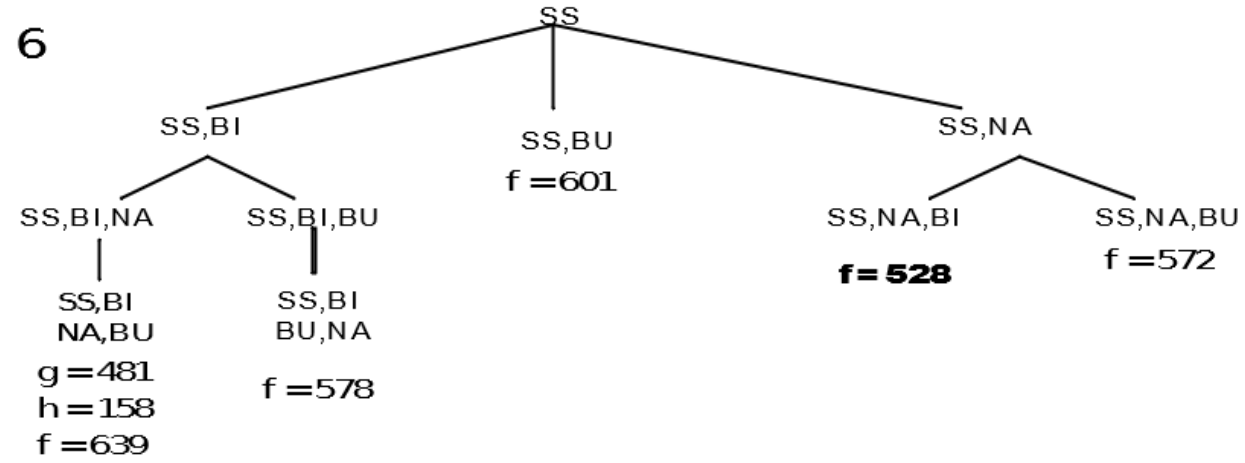


Ramificar y acotar v2

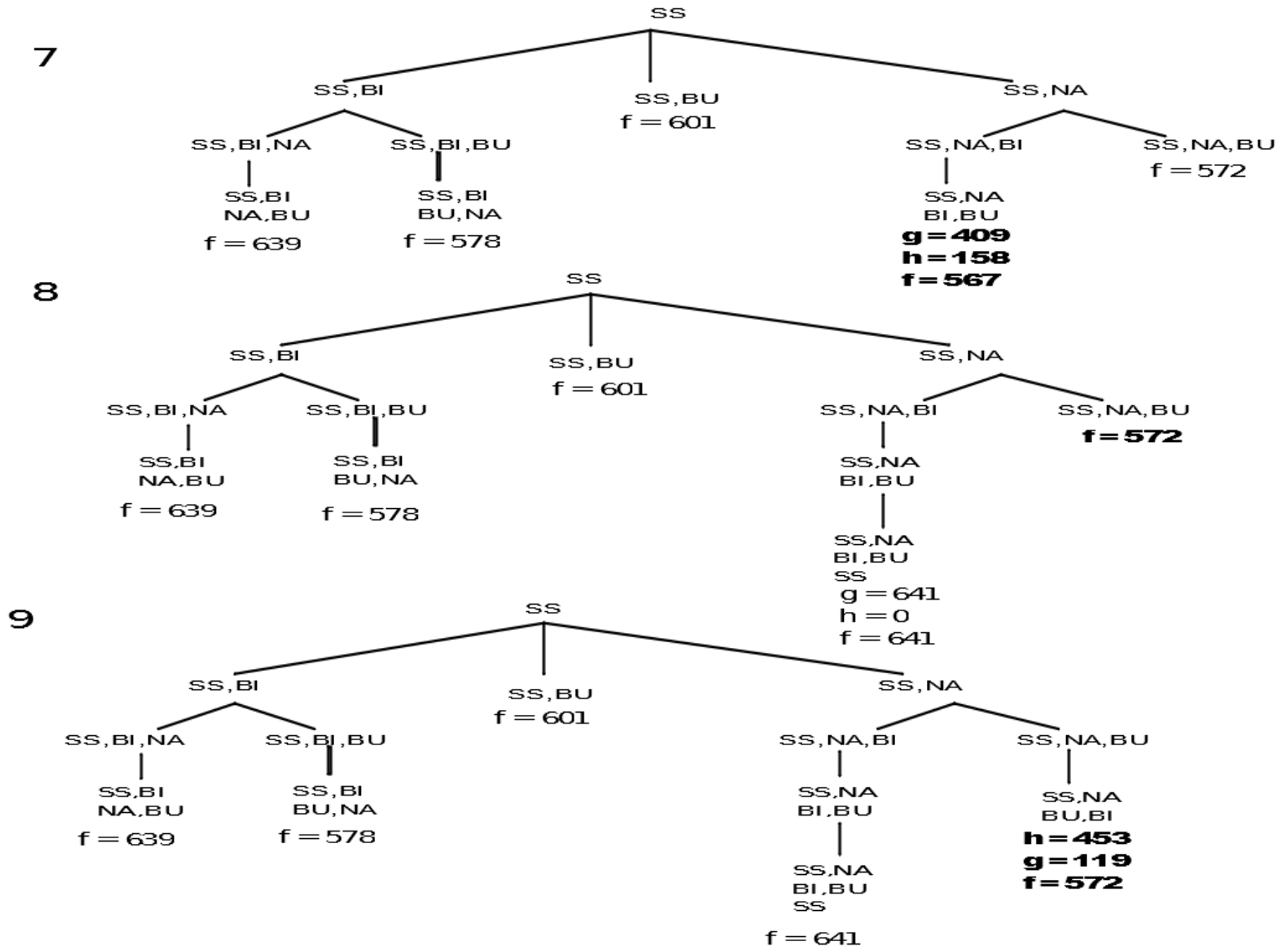
5



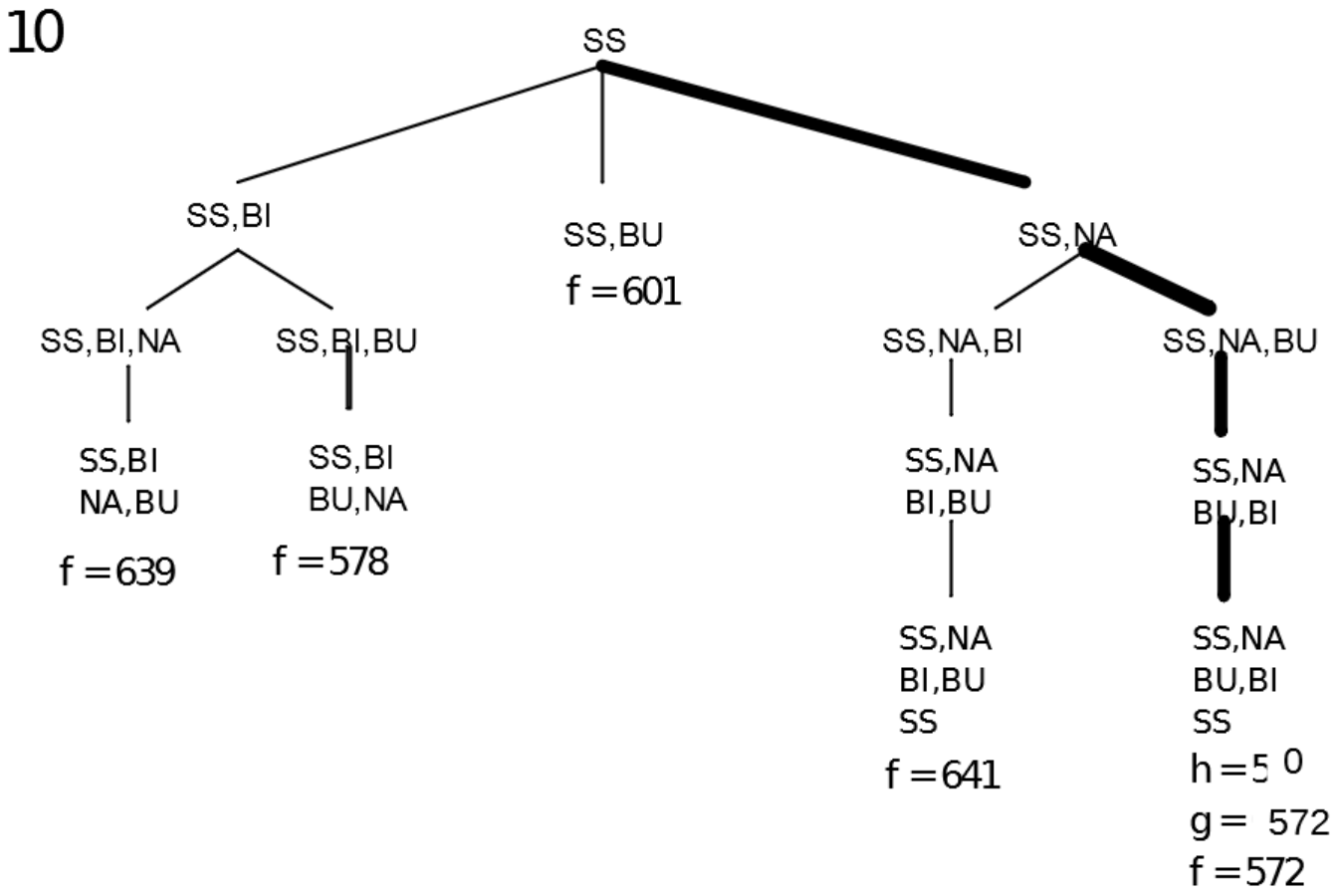
6



Ramificar y acotar v2



Ramificar y acotar v2



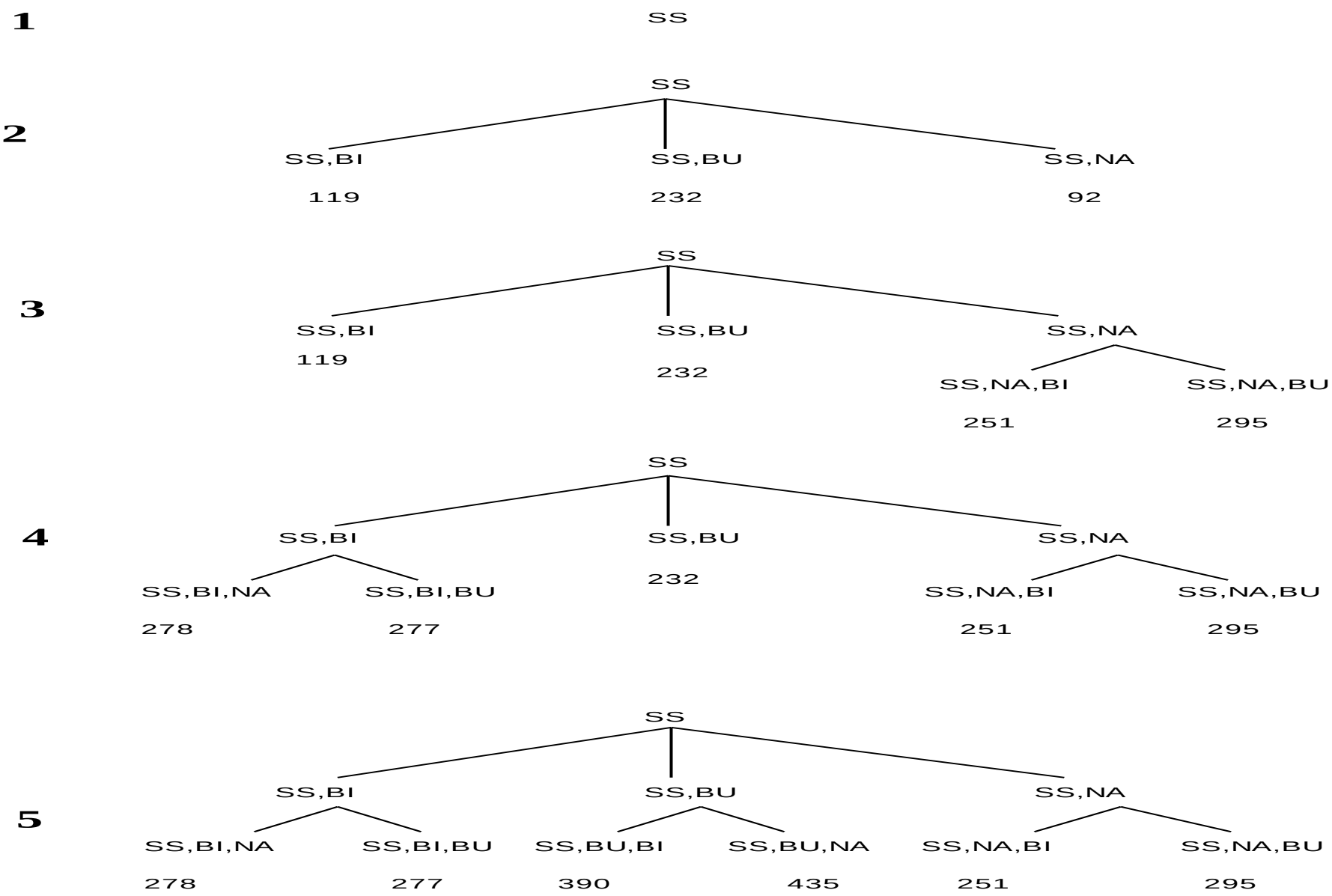
Ramificar y acotar v3

➢ Algoritmo

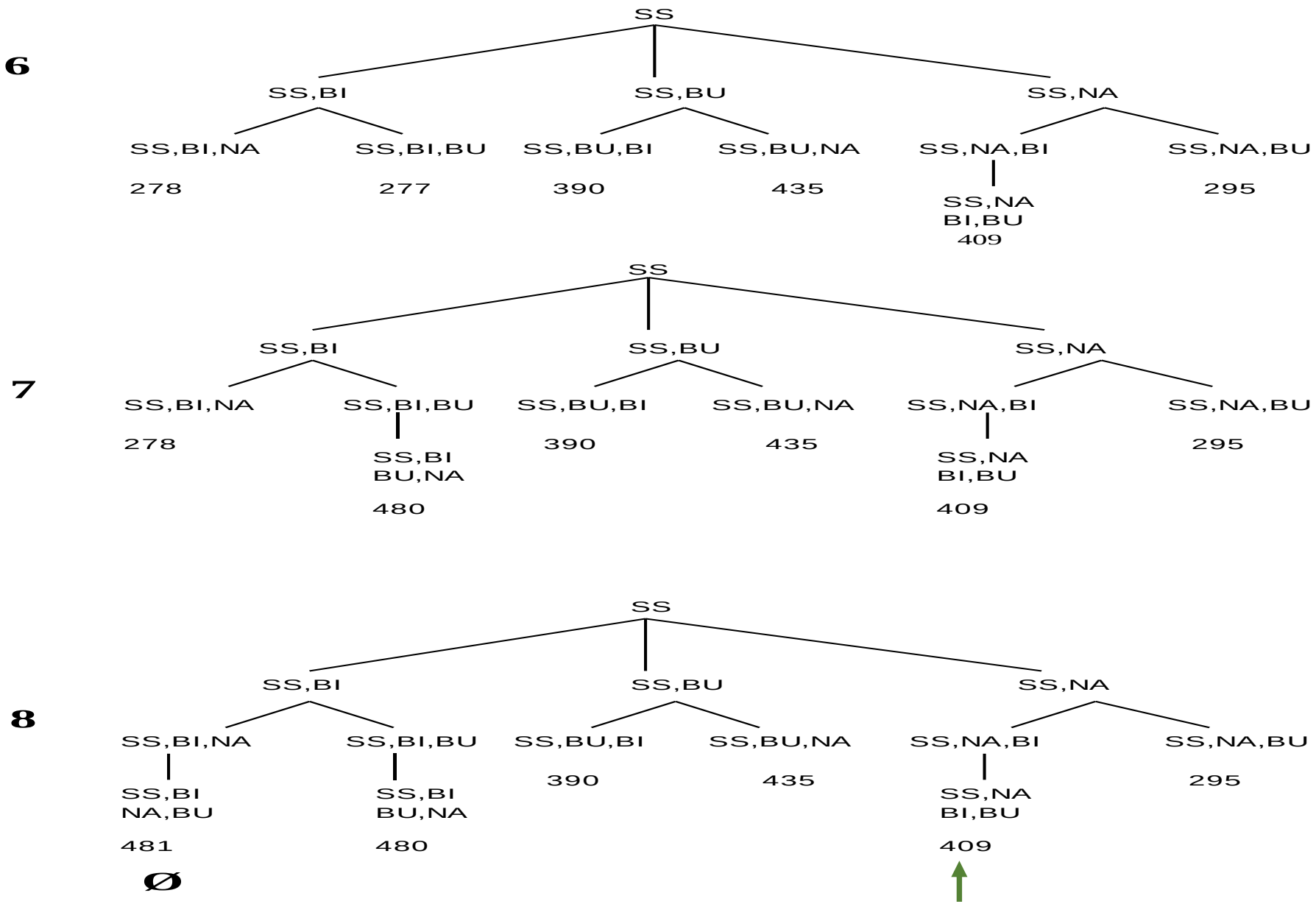
- Construir una lista de caminos parciales con el nodo raíz (**coste acumulado = g = 0**).
- Hasta que la lista esté vacía o (el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino)
  - Eliminar primer camino de la lista
  - Formar nuevos caminos a partir del eliminado añadiendo tanto caminos como hijos tenga el último nodo de ese camino
  - Añadir esos nuevos caminos a la lista junto con su coste
  - Ordenar la lista de menor a mayor considerando la función del coste en ese momento
  - Si hay **CAMINOS REPETIDOS**, borrar todos ellos, excepto aquel que alcanza el nodo con el coste mínimo.

➢ Si el primer camino de la lista encuentra el nodo objetivo, anunciar éxito, sino, fallo.

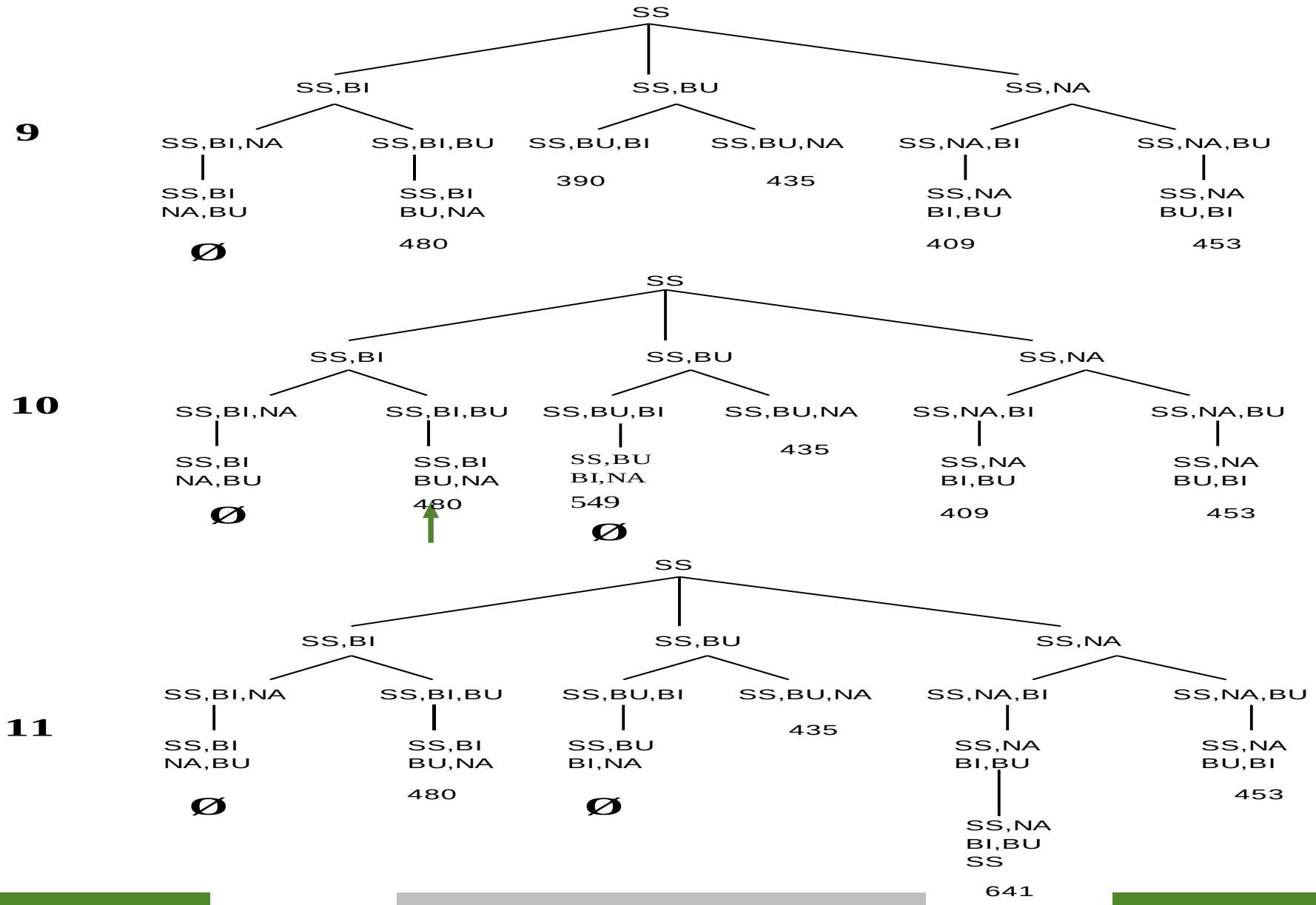
Ramificar y acotar v3



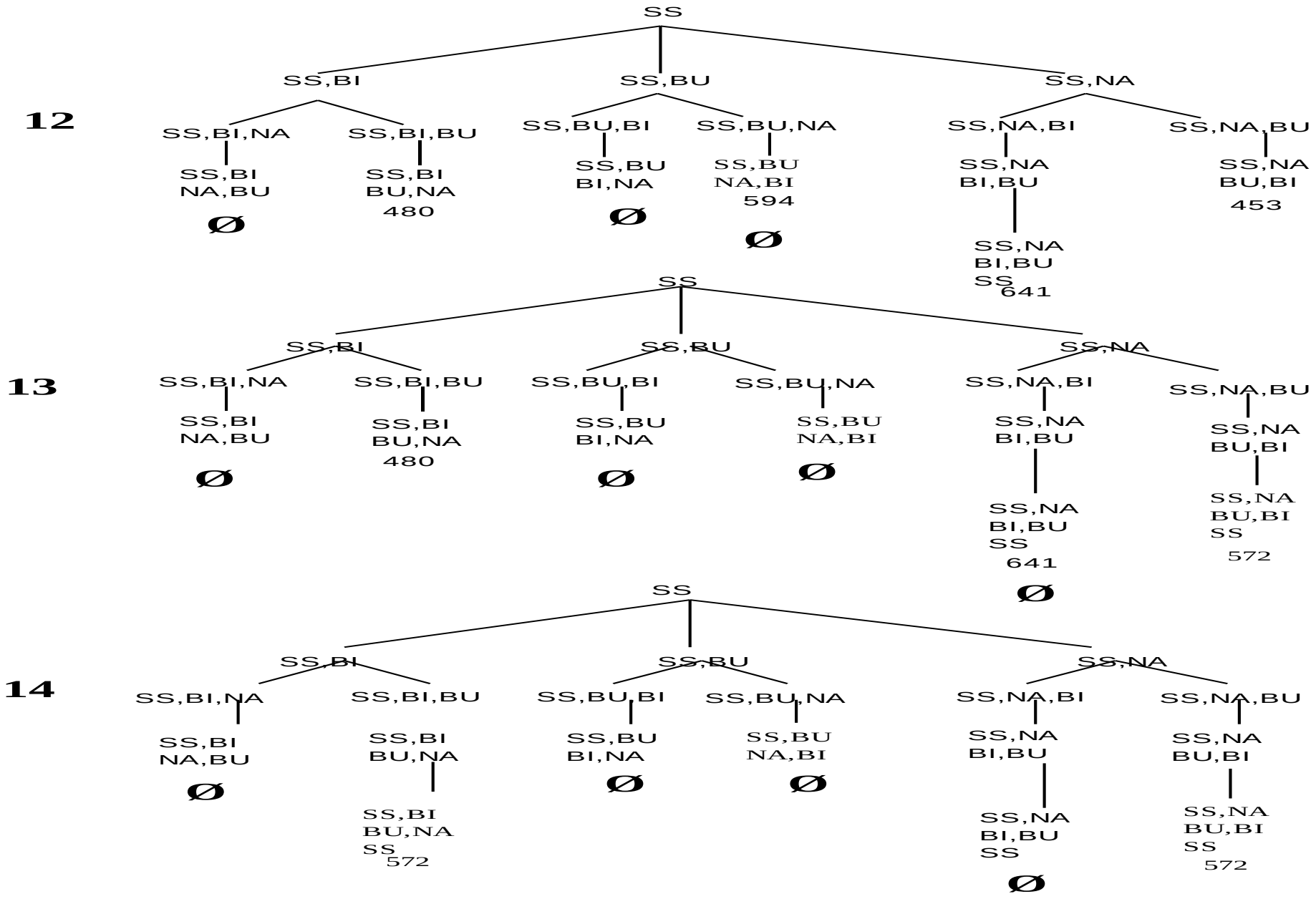
Ramificar y acotar v3



Ramificar y acotar v3



Ramificar y acotar v3



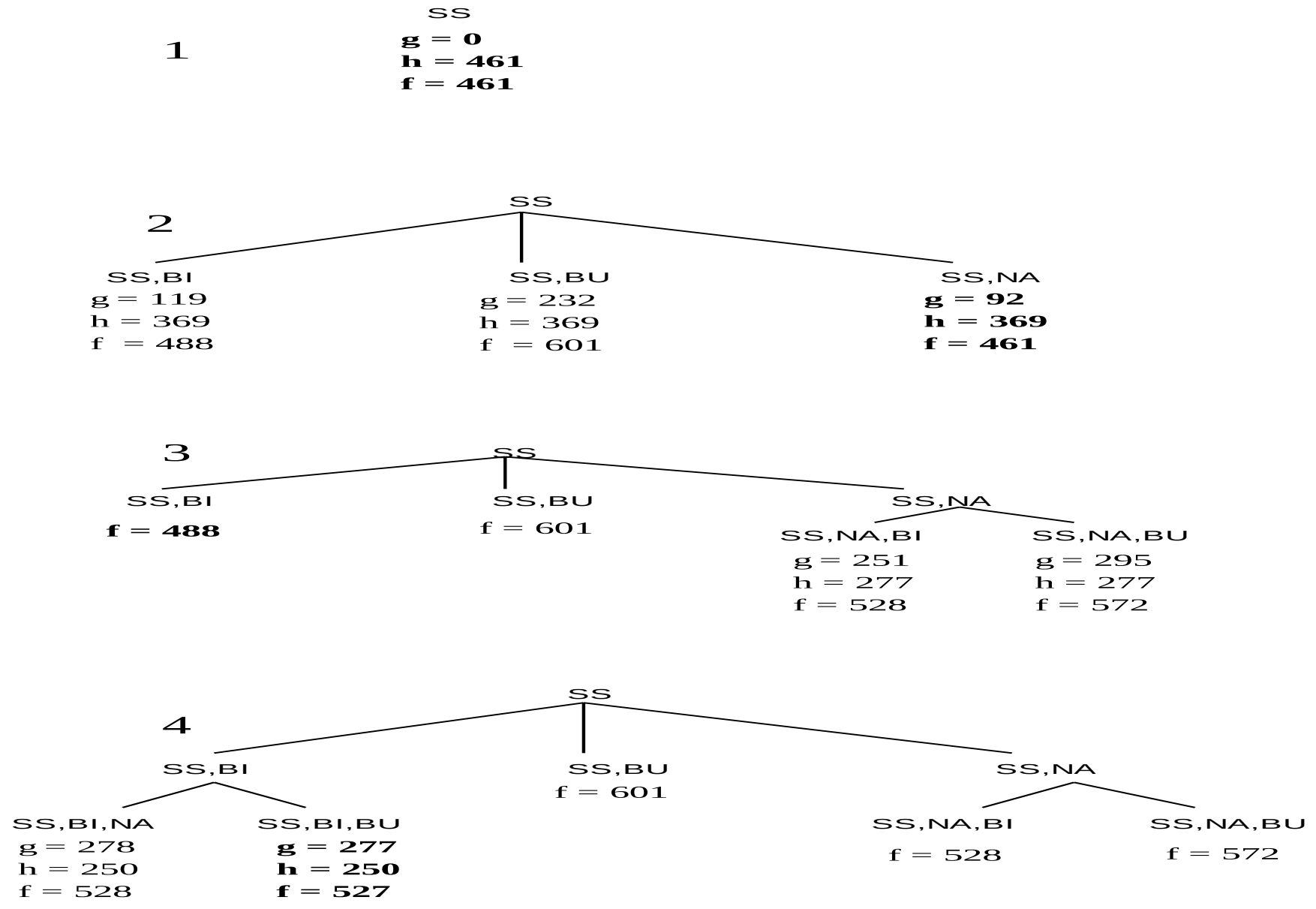
Ramificar y acotar v4

➢ Algoritmo

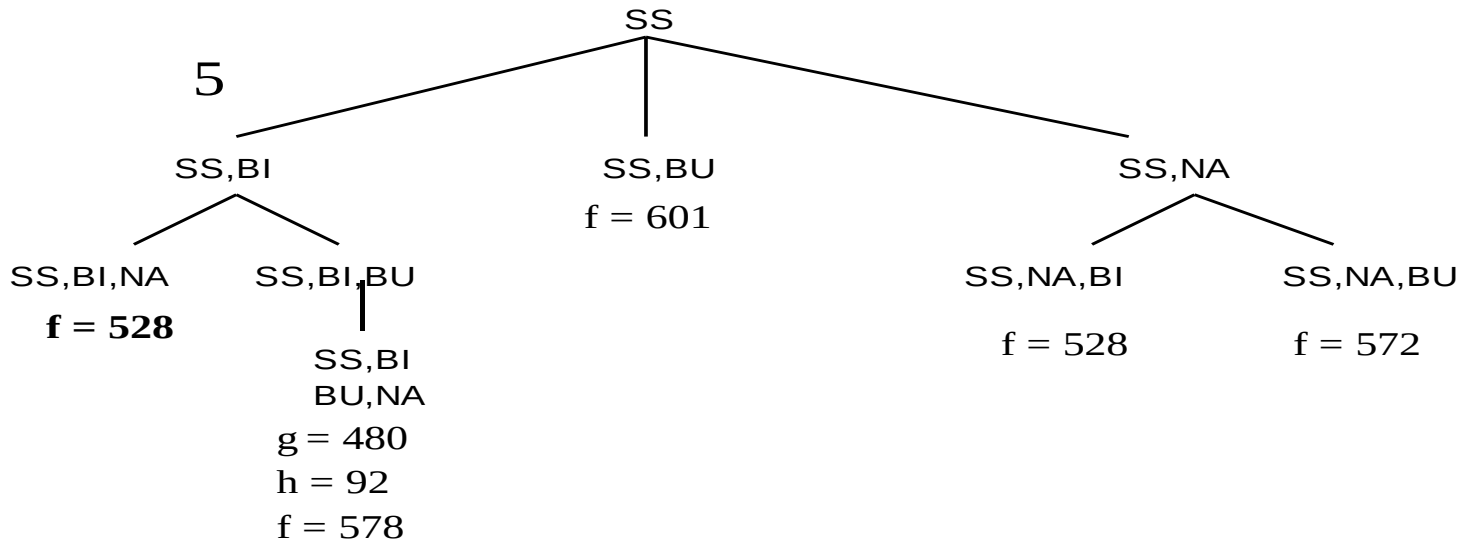
- Construir una lista de caminos parciales con el nodo raíz (***f = g + h*** ; ***g***=coste acumulado, ***h***=coste heurístico hasta la solución).
- Hasta que la lista esté vacía o (el camino alcance el nodo objetivo y el coste del camino <= que el coste de cualquier otro camino)
  - Eliminar primer camino de la lista
  - Formar nuevos caminos a partir del eliminado añadiendo tanto caminos como hijos tenga el último nodo de ese camino
  - Añadir esos nuevos caminos a la lista junto con su **coste total (*f*)**
  - Ordenar la lista de menor a mayor considerando la función del coste en ese momento
  - Si hay **CAMINOS REPETIDOS**, borrar todos ellos, excepto aquel que alcanza el nodo con el coste mínimo.

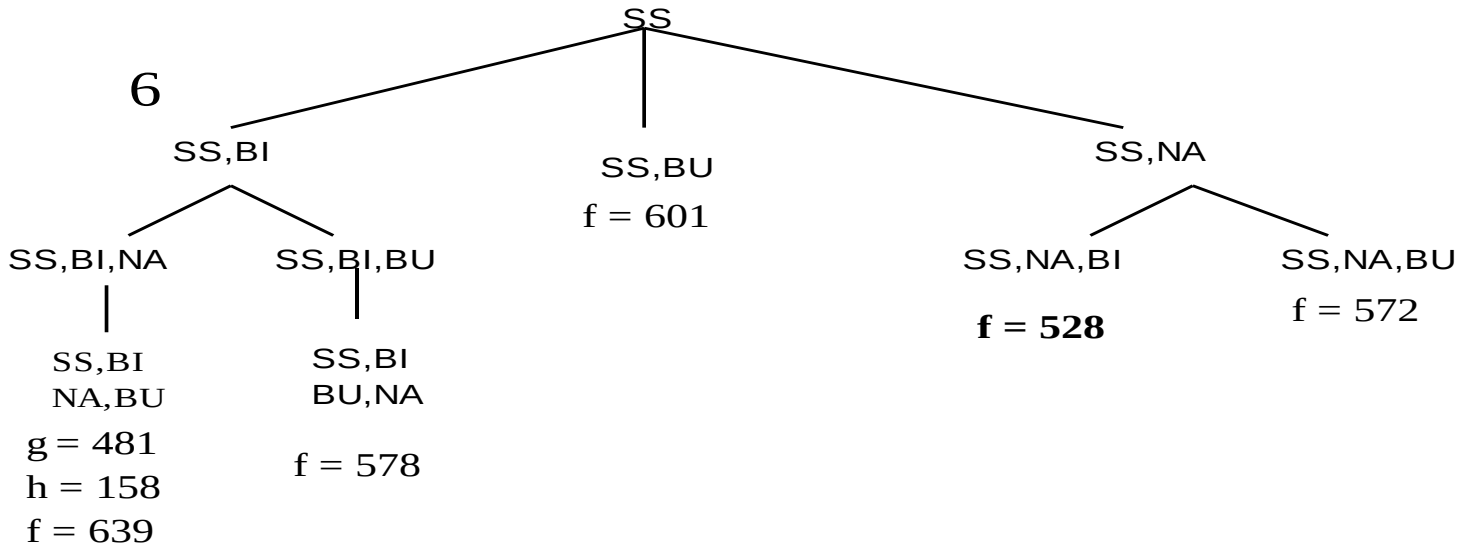
➢ Si el primer camino de la lista encuentra el nodo objetivo, anunciar éxito, sino, fallo.

Ramificar y acotar v4

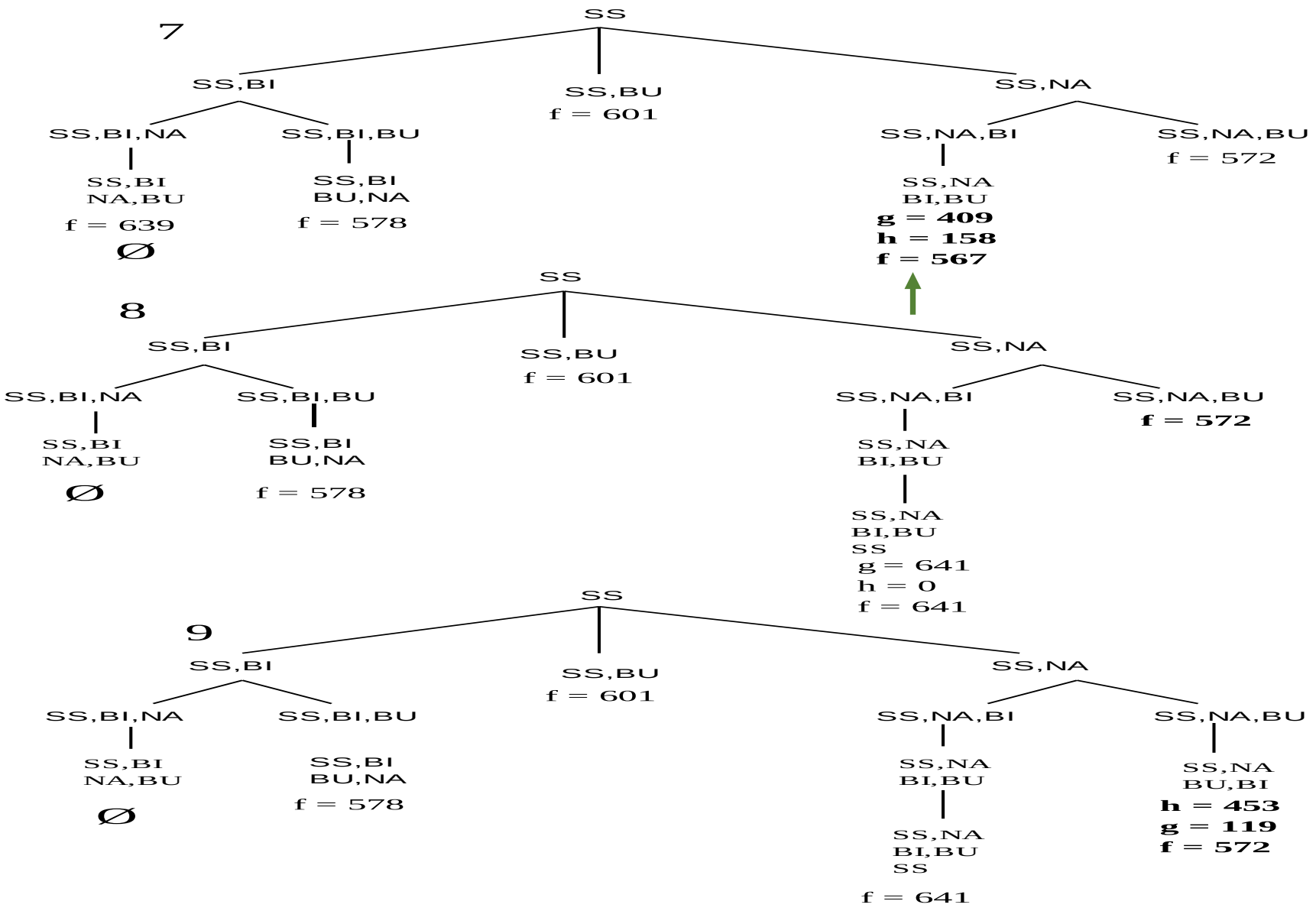


Ramificar y acotar v4

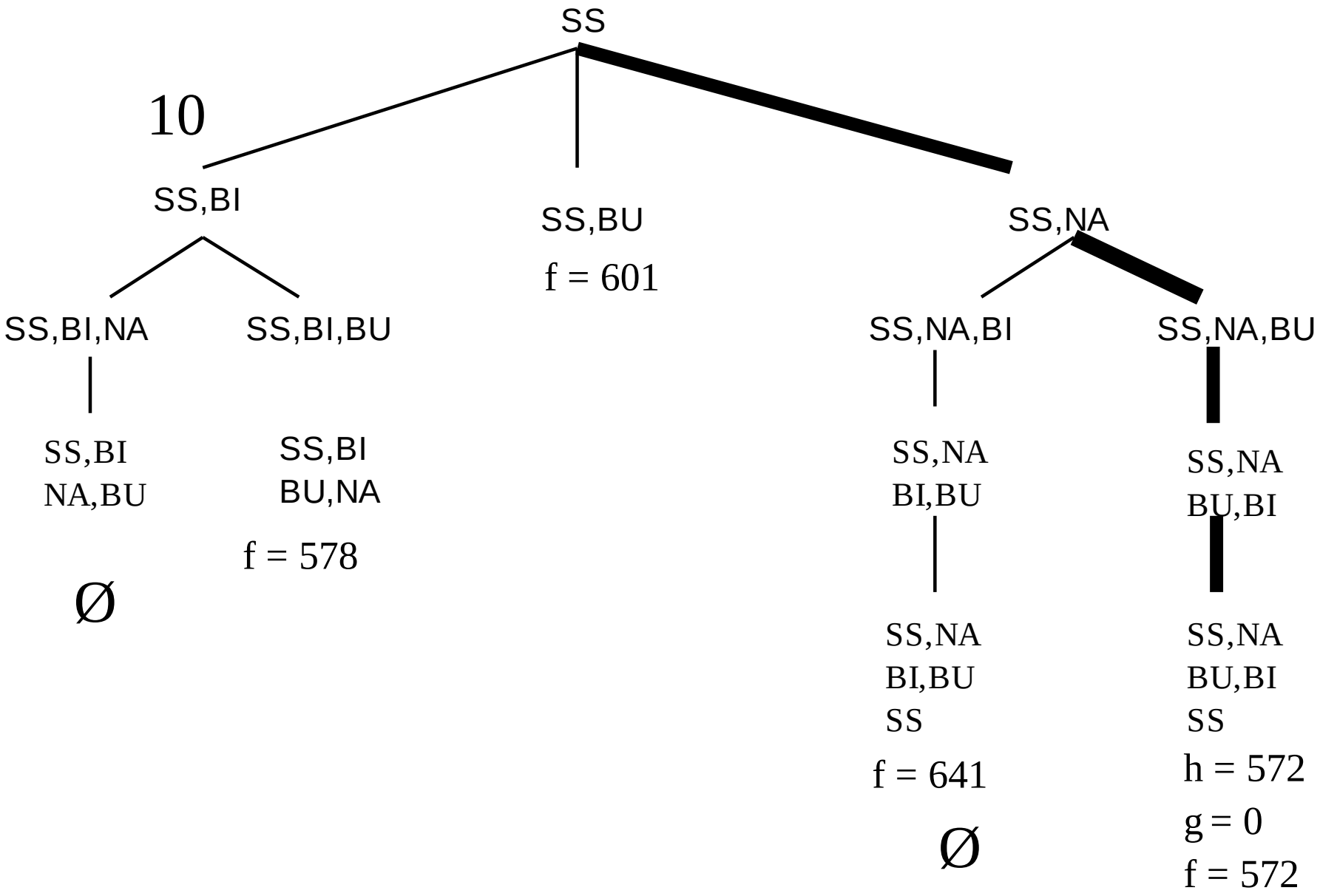




Ramificar y acotar v4

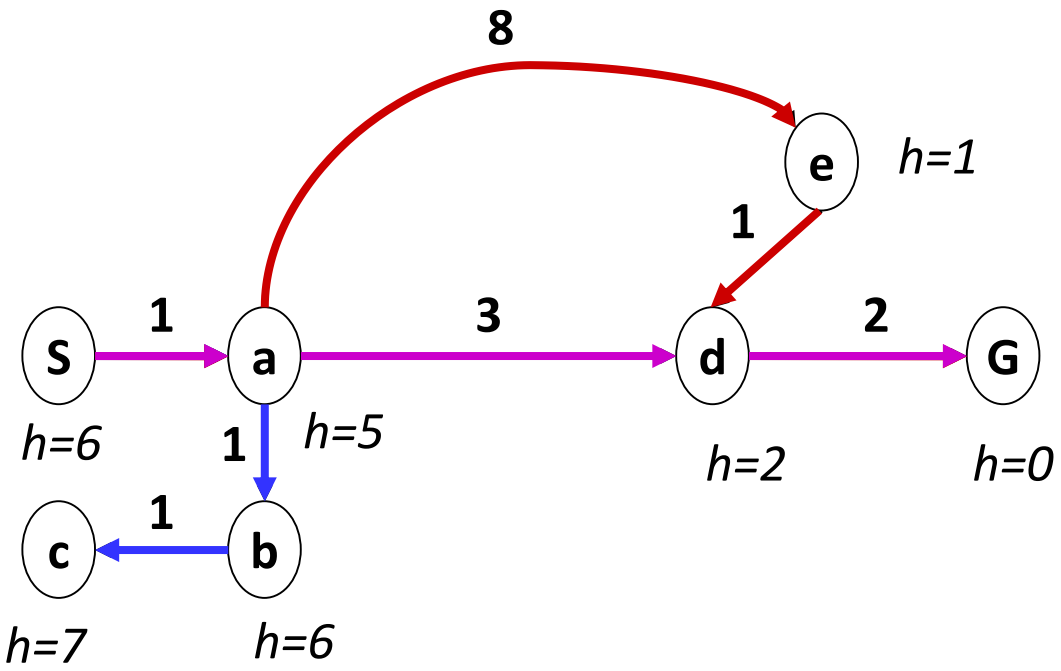


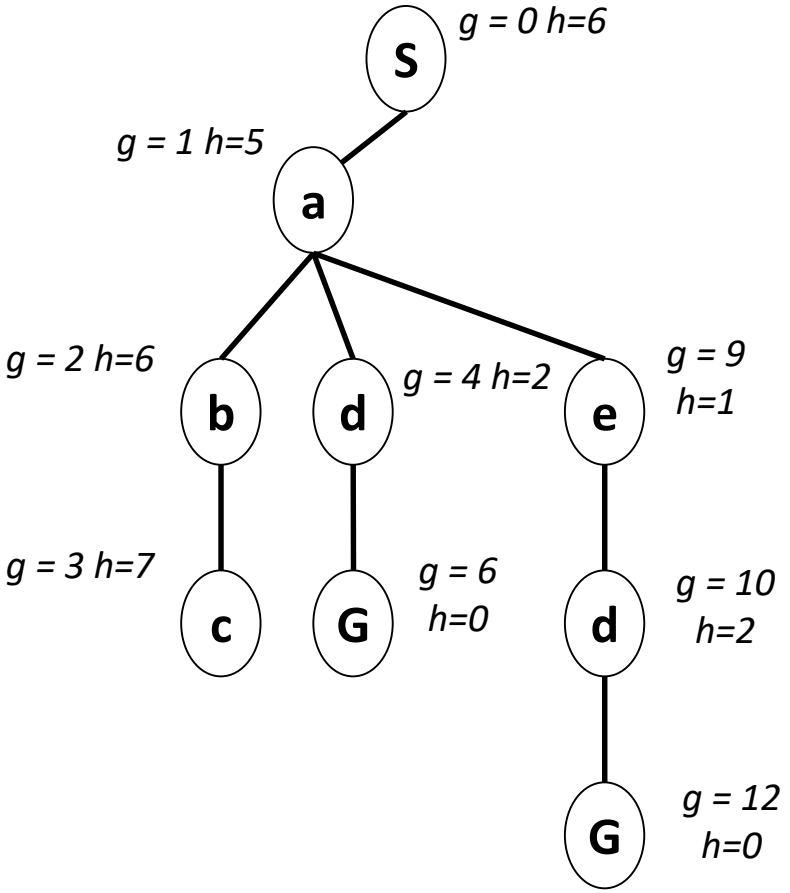
Ramificar y acotar v4



Combinando UCS y Greedy

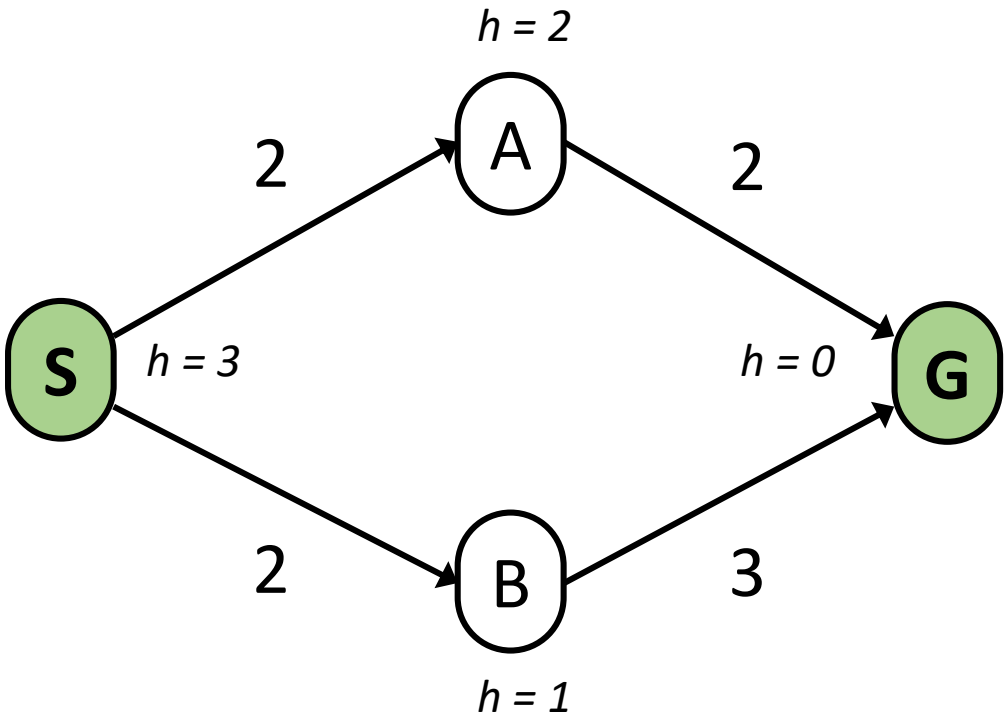
- Coste uniforme (UCS) ordena por **el coste del camino**, o backward cost **g(n)**
- Greedy ordena por **la cercanía del objetivo**, o forward cost **h(n)**
- A* Search ordena por la suma: **f(n) = g(n) + h(n)**





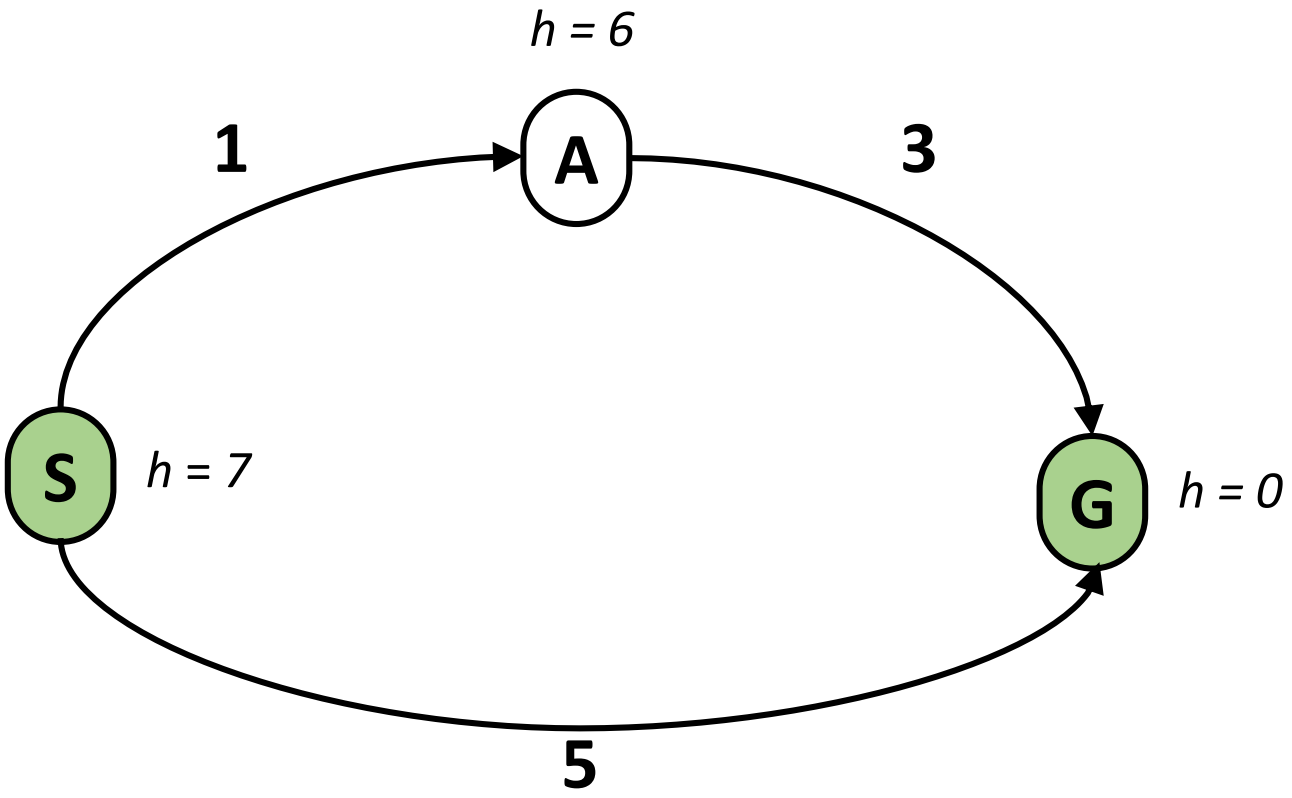
¿Cuándo debería terminar A*?

- ¿Cuando encolamos un objetivo?



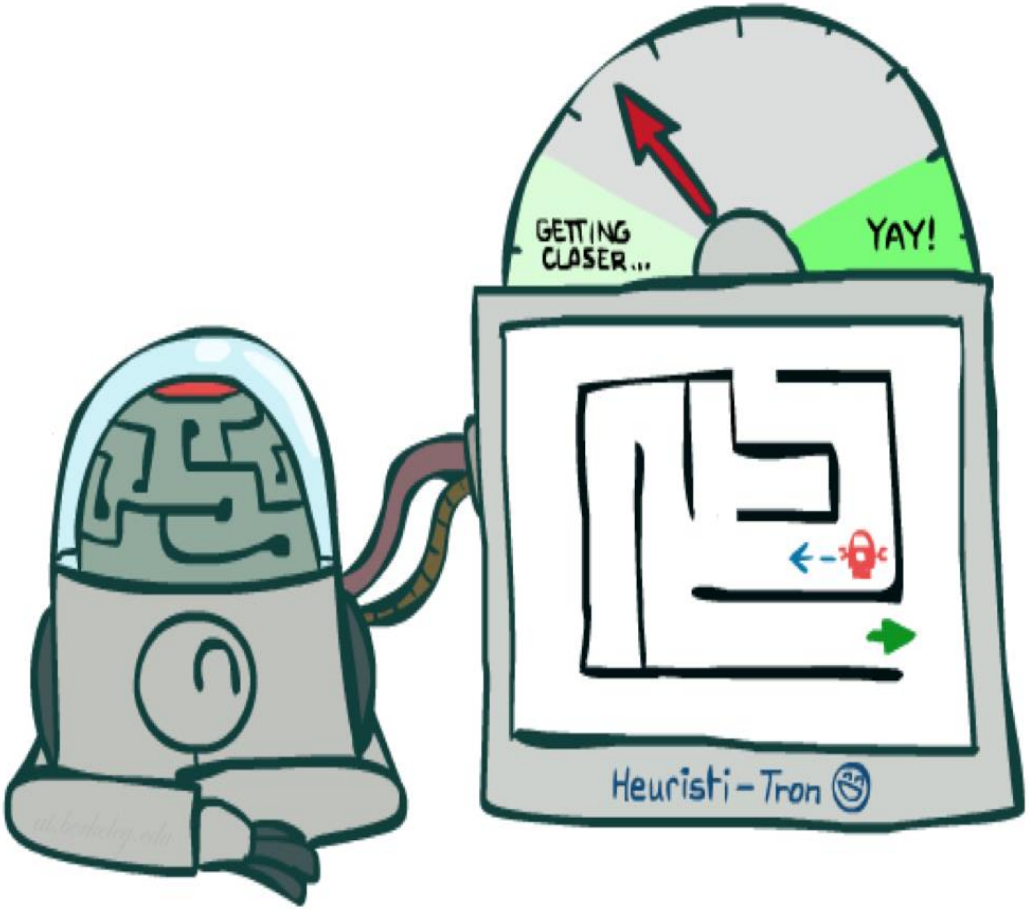
- **No**: solo se debe parar cuando desencolamos un objetivo

¿Es A* Óptimo?

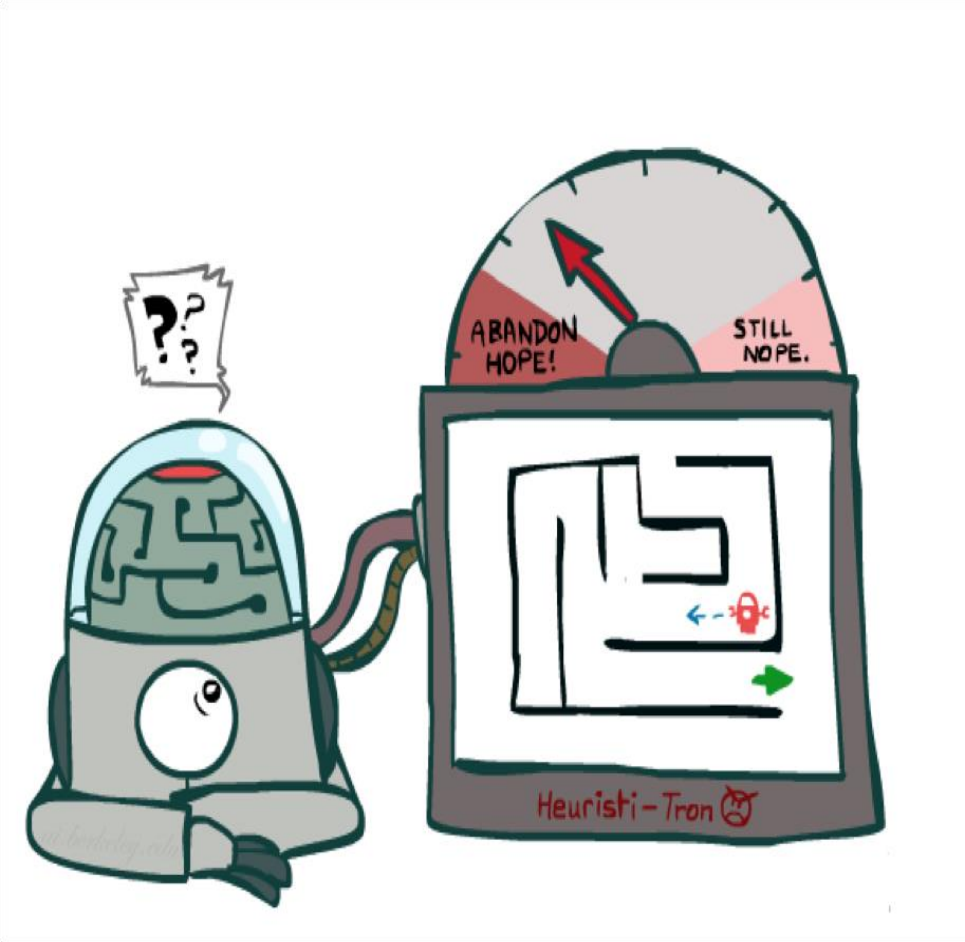


- ¿Qué es lo que falla?
- **¡Necesitamos que las estimaciones sean menores que los costes reales!**

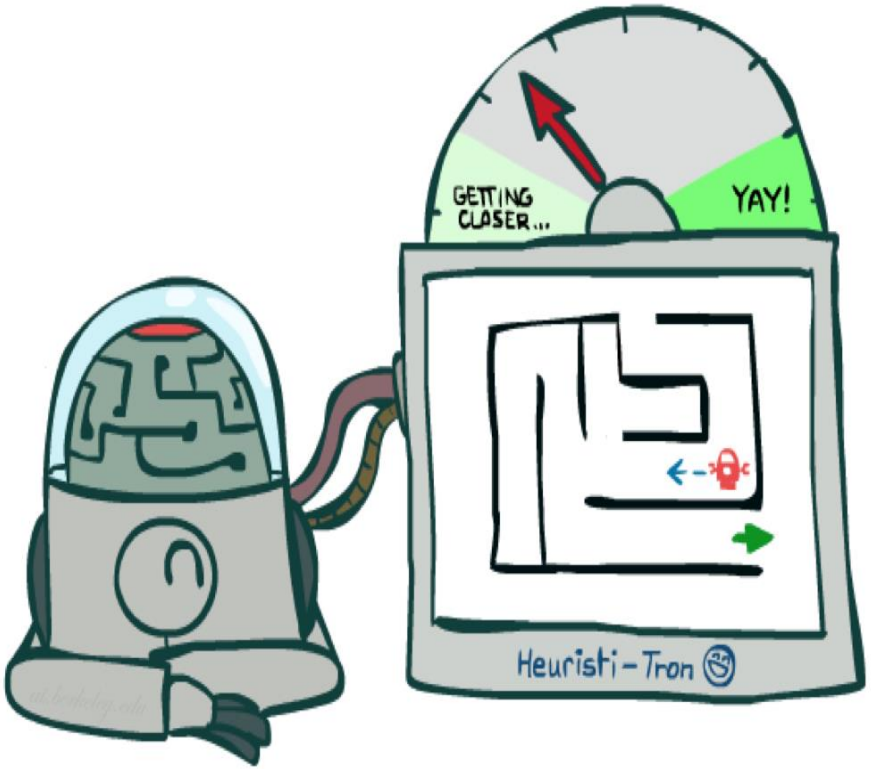
Heurísticos admisibles



Idea: admisibilidad



Heurísticos **inadmisibles** (pesimistas) rompen la **optimalidad** al *atrapar* buenos planes en el borde



Los heurísticos **admisibles** (optimistas) ralentizan el proceso (planes malos) pero **nunca superan el coste real**

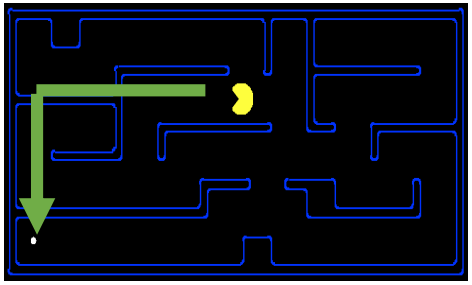
Heurísticos admisibles

- Un heurístico h es **admisible** (optimista) si:

$$0 \leq h(n) \leq h^*(n)$$

donde $h^*(n)$ es el coste real al objetivo más cercano

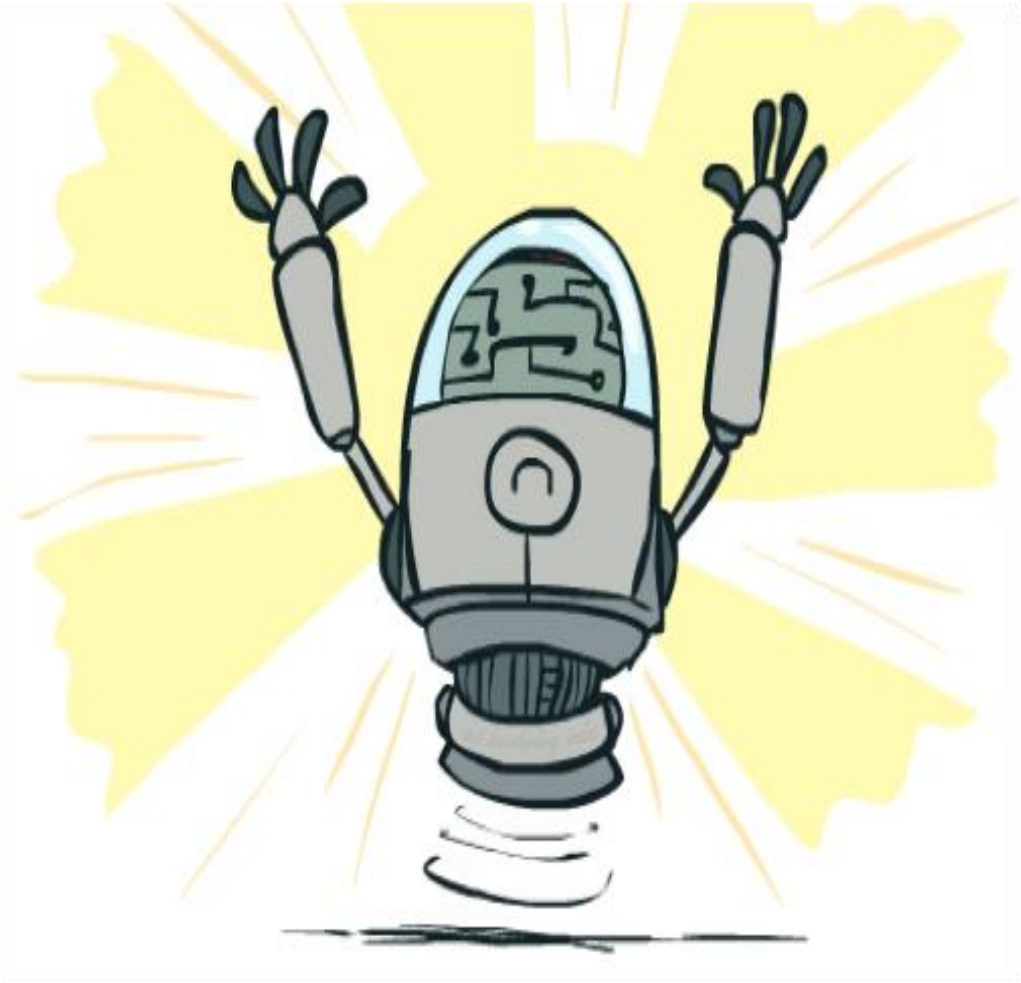
- Ejemplos:



15

- Encontrar heurísticos **admisibles** es una de las partes más importantes al usar A* en la práctica

Optimalidad de A* en Tree Search



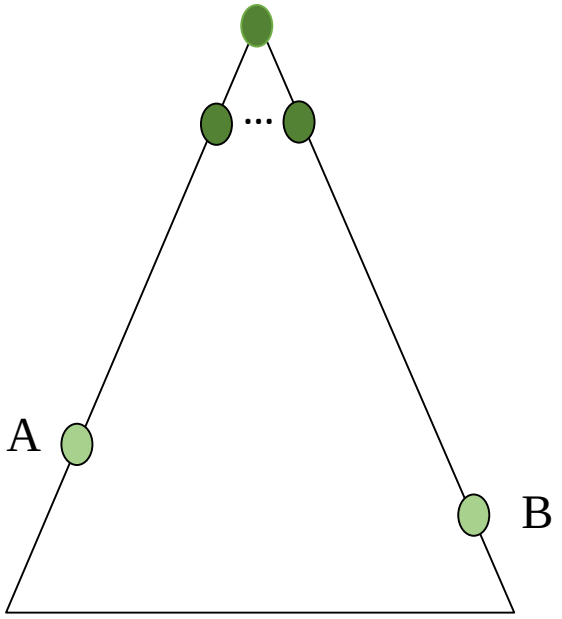
Optimalidad de A* en Tree Search

Asumimos:

- **A** es un nodo **objetivo óptimo**
- **B** es un nodo **objetivo subóptimo**
- ***h*** es **admisible**

Proposición:

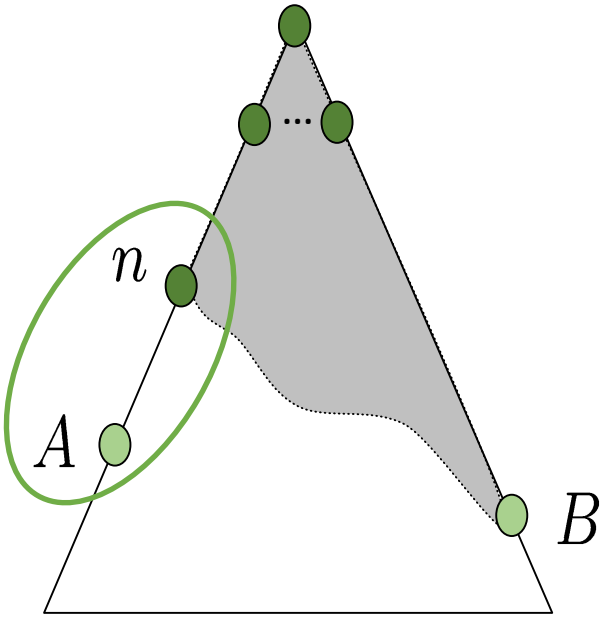
- **A** saldrá del borde antes que **B**



Optimalidad de A* en Tree Search: Bloqueo

Prueba:

- Imaginemos que **B** está en el borde
- Algún antecesor **n** de **A** está en el borde, (¡también puede ser **A**!)
- Proposición: **n** será expandido antes que **B**
  - **f(n) es menor o igual que f(A)**



$\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(n)$ Definición de coste-f

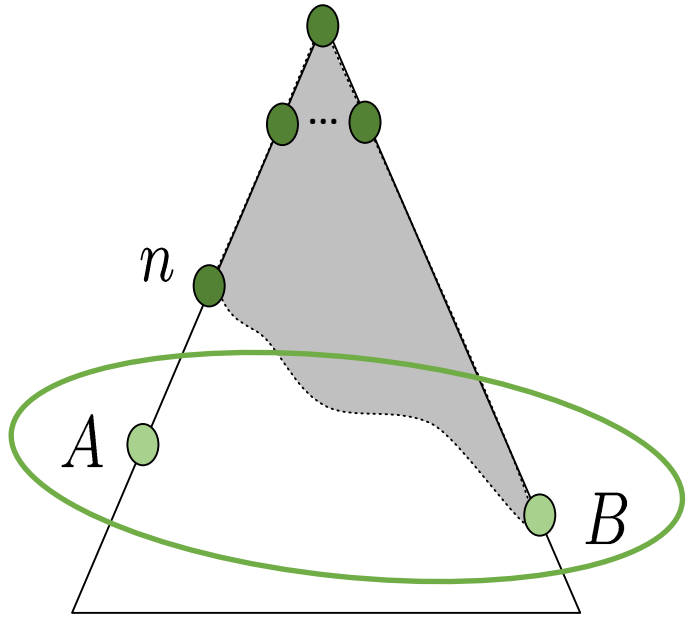
$\mathbf{f}(n) \leq \mathbf{g}(A)$ Admisibilidad de h

$\mathbf{g}(A) = \mathbf{f}(A)$ h = 0 en el objetivo

Optimalidad de A* en Tree Search: Bloqueo

Prueba:

- Imaginemos que **B** está en el borde
- Algún antecesor **n** de **A** está en el borde, (¡también puede ser **A**!)
- Proposición: **n** será expandido antes que **B**
  - **f(n)** es menor o igual que **f(A)**
  - **f(A)** es menor que **f(B)**



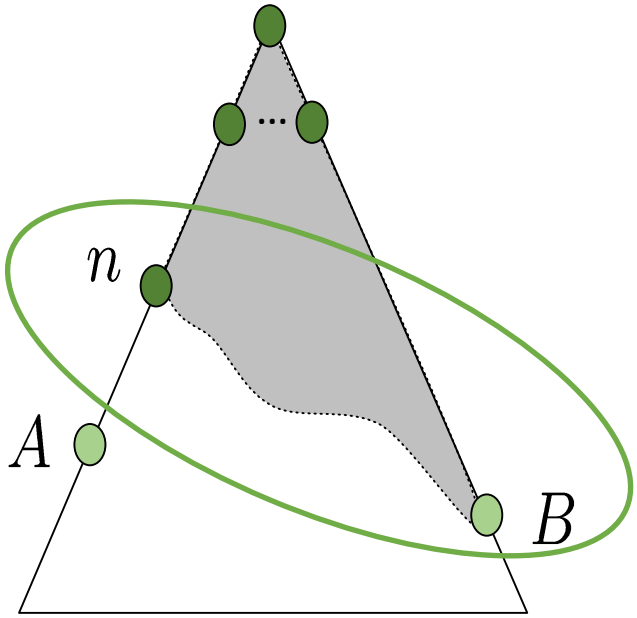
$\mathbf{g}(A) < \mathbf{g}(B)$ **B** es subóptimo

$\mathbf{f}(A) < \mathbf{f}(B)$ $h = 0$ en un objetivo

Optimalidad de A* en Tree Search: Bloqueo

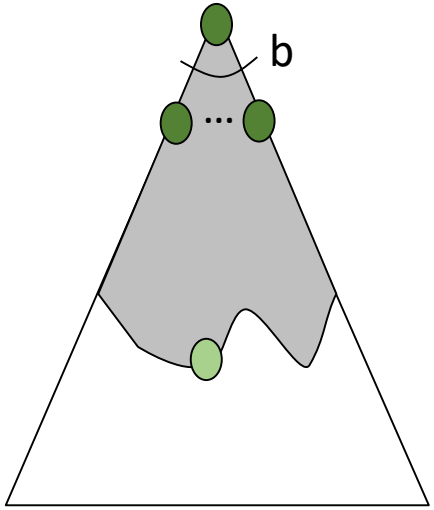
Prueba:

- Imaginemos que **B** está en el borde
- Algún antecesor **n** de **A** está en el borde, (¡también puede ser **A**!)
- Proposición: **n** será expandido antes que **B**
  - **f(n)** es menor o igual que **f(A)**
  - **f(A)** es menor que **f(B)**
  - **n** se expande antes que **B**
- Todos los antecesores de **A** se expanden antes que **B**
- **A** se expande antes que **B**
- **A*** search es óptima

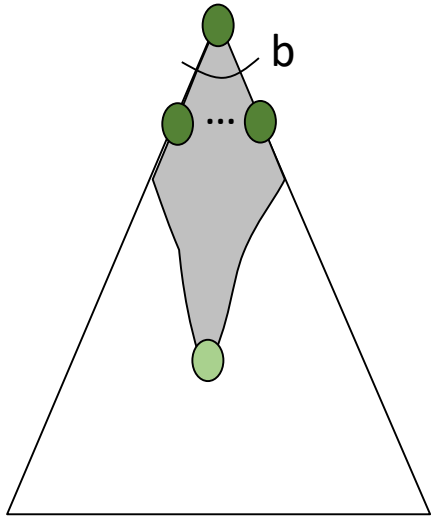


$$f(n) \leq f(A) < f(B)$$

Propiedades de A*



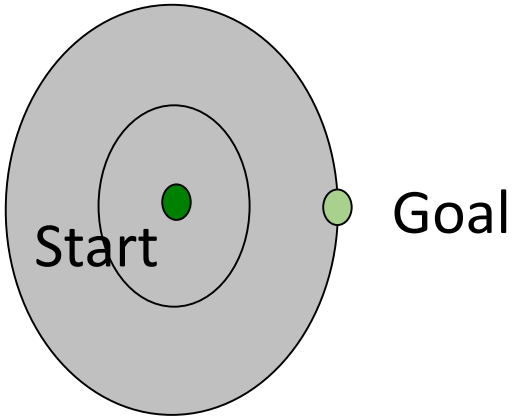
Coste-Uniforme



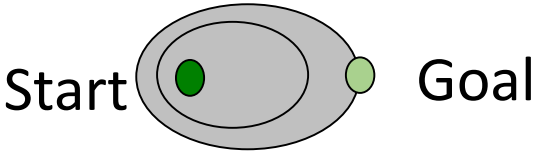
A*

UCS vs A*

- Coste-Uniforme se expande igualmente en todas direcciones

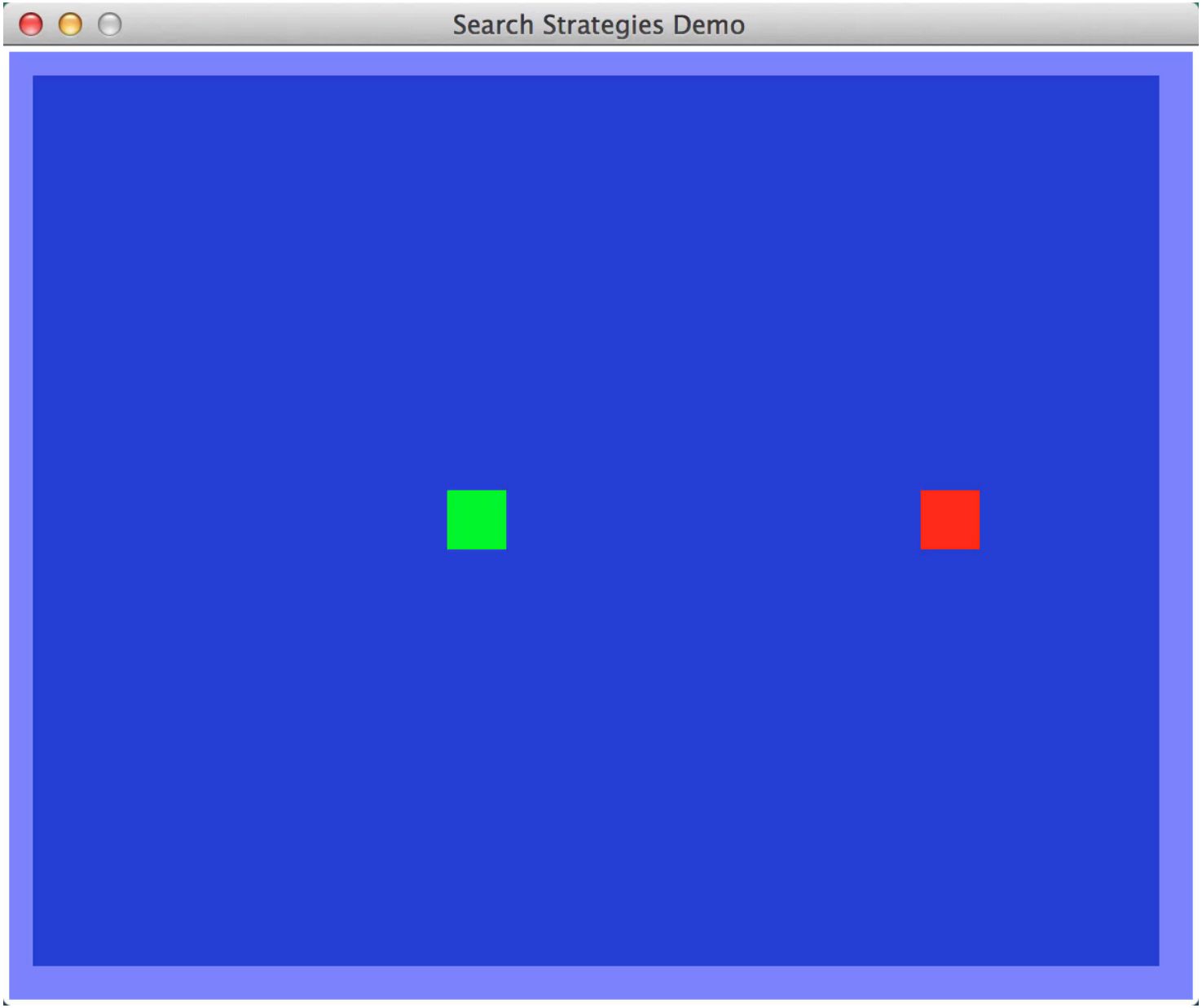


- A* se expande principalmente hacia el objetivo, pero guarda sus opciones para asegurar la optimalidad

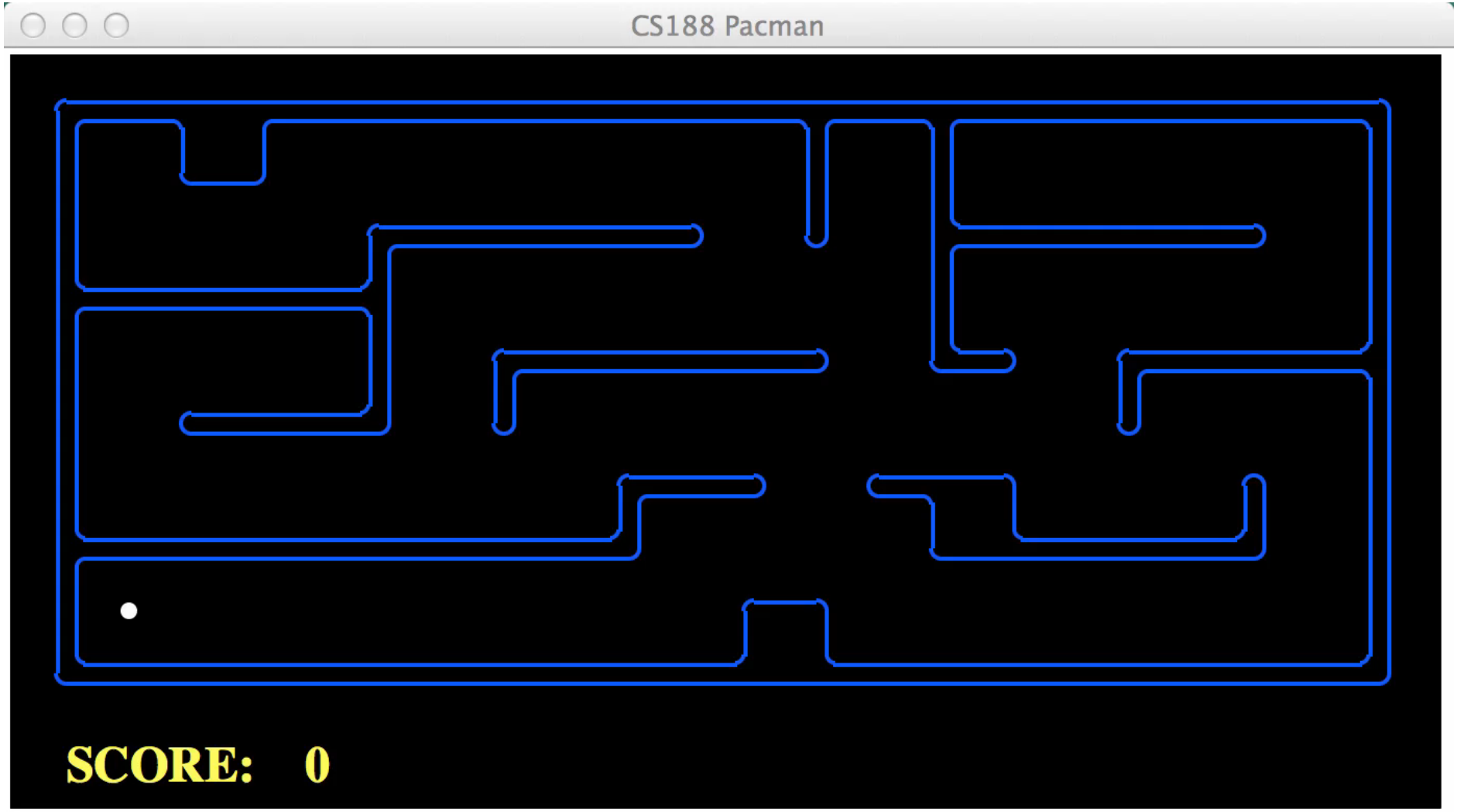


[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

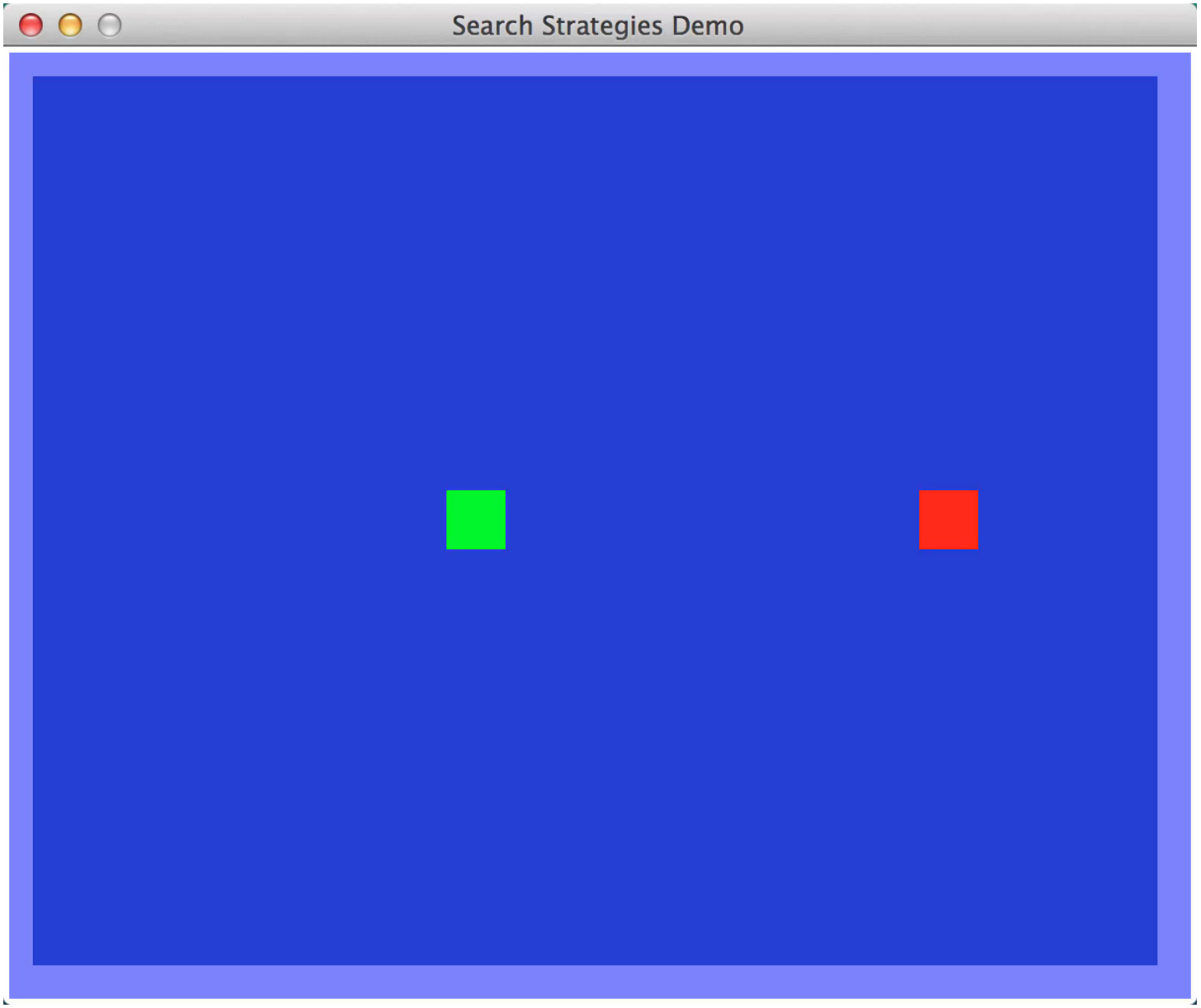
Vídeo of Demo Contours (Empty) -- UCS



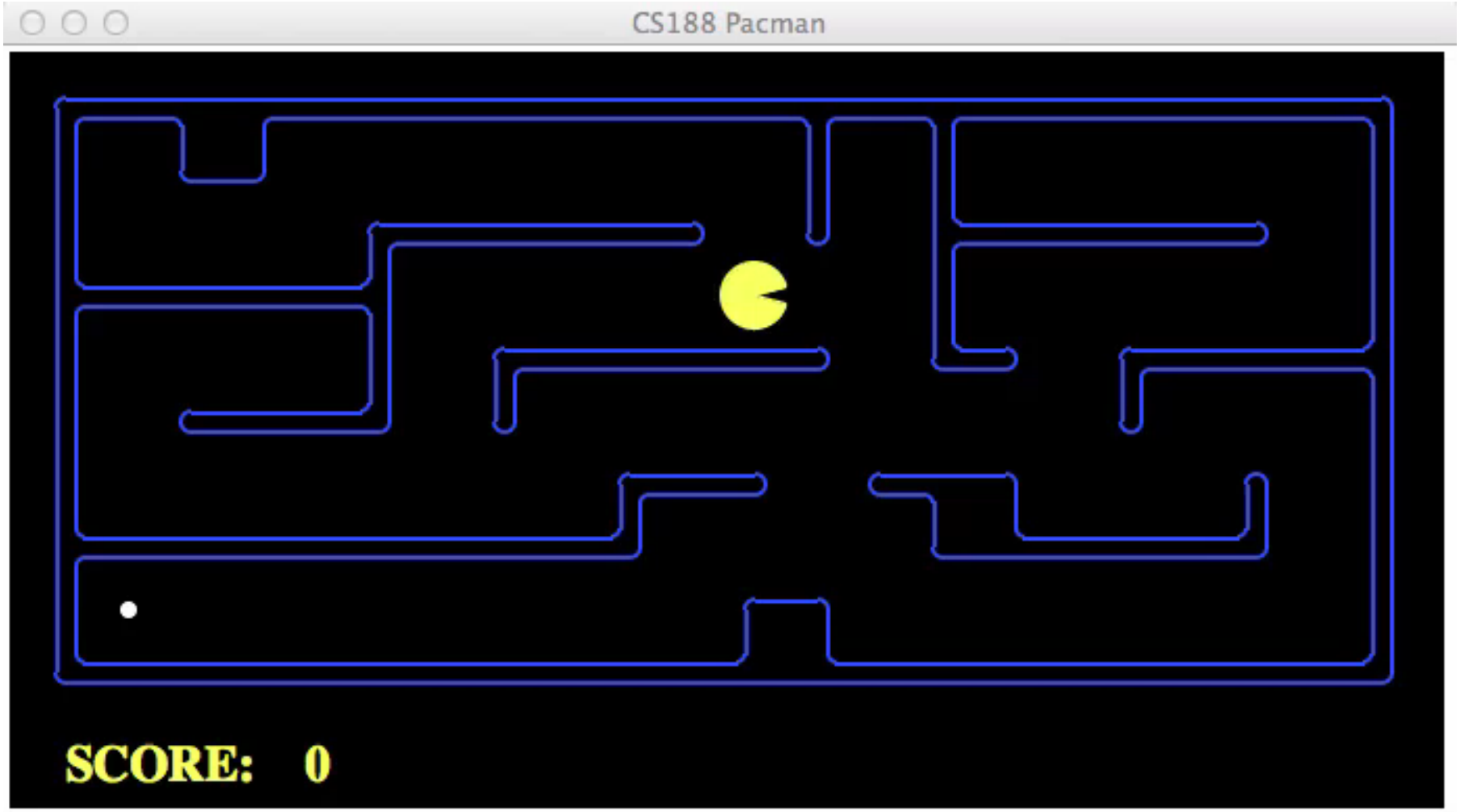
Vídeo of Demo Contours (Pacman) -- UCS



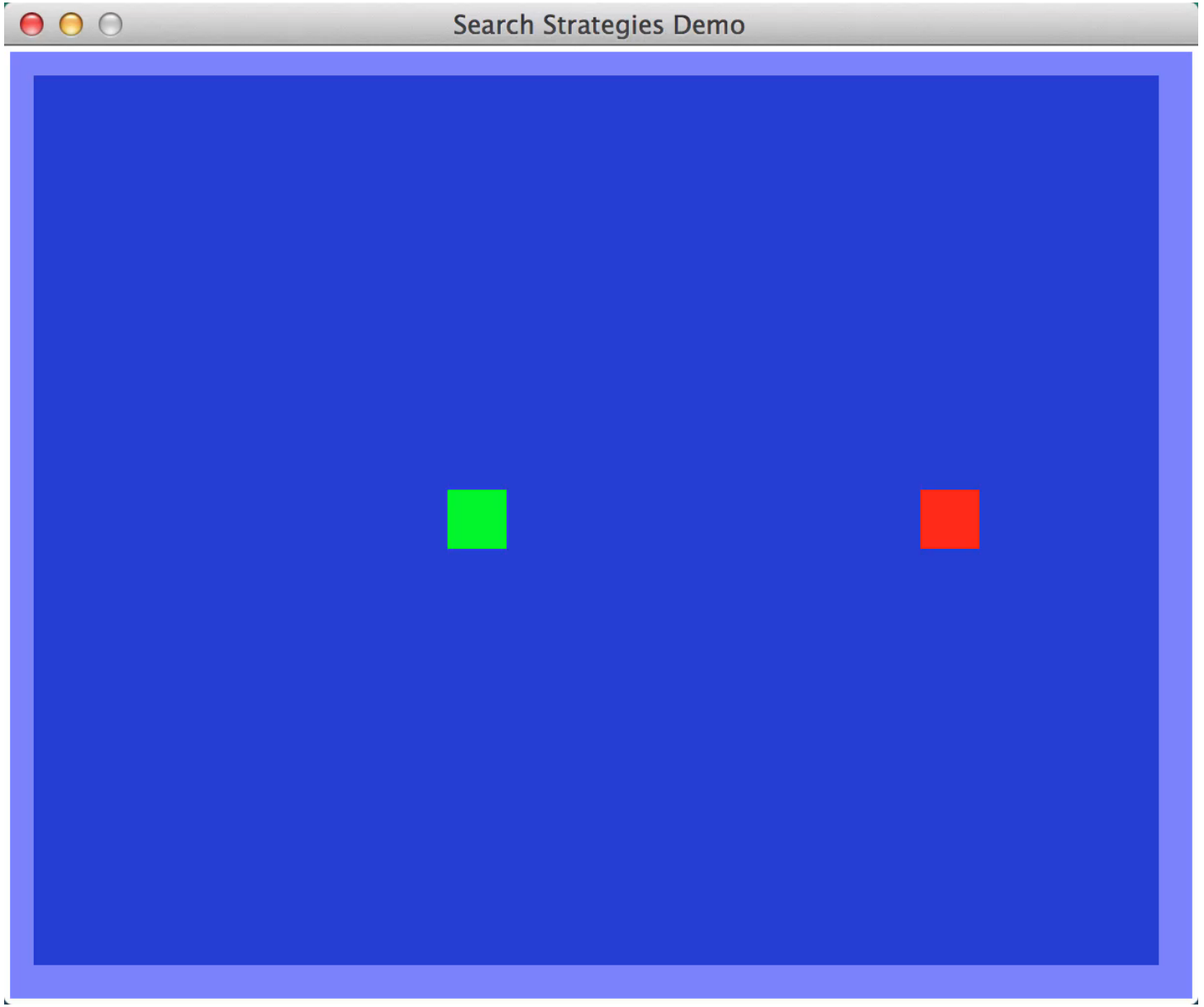
Vídeo of Demo Contours (Empty) -- Greedy



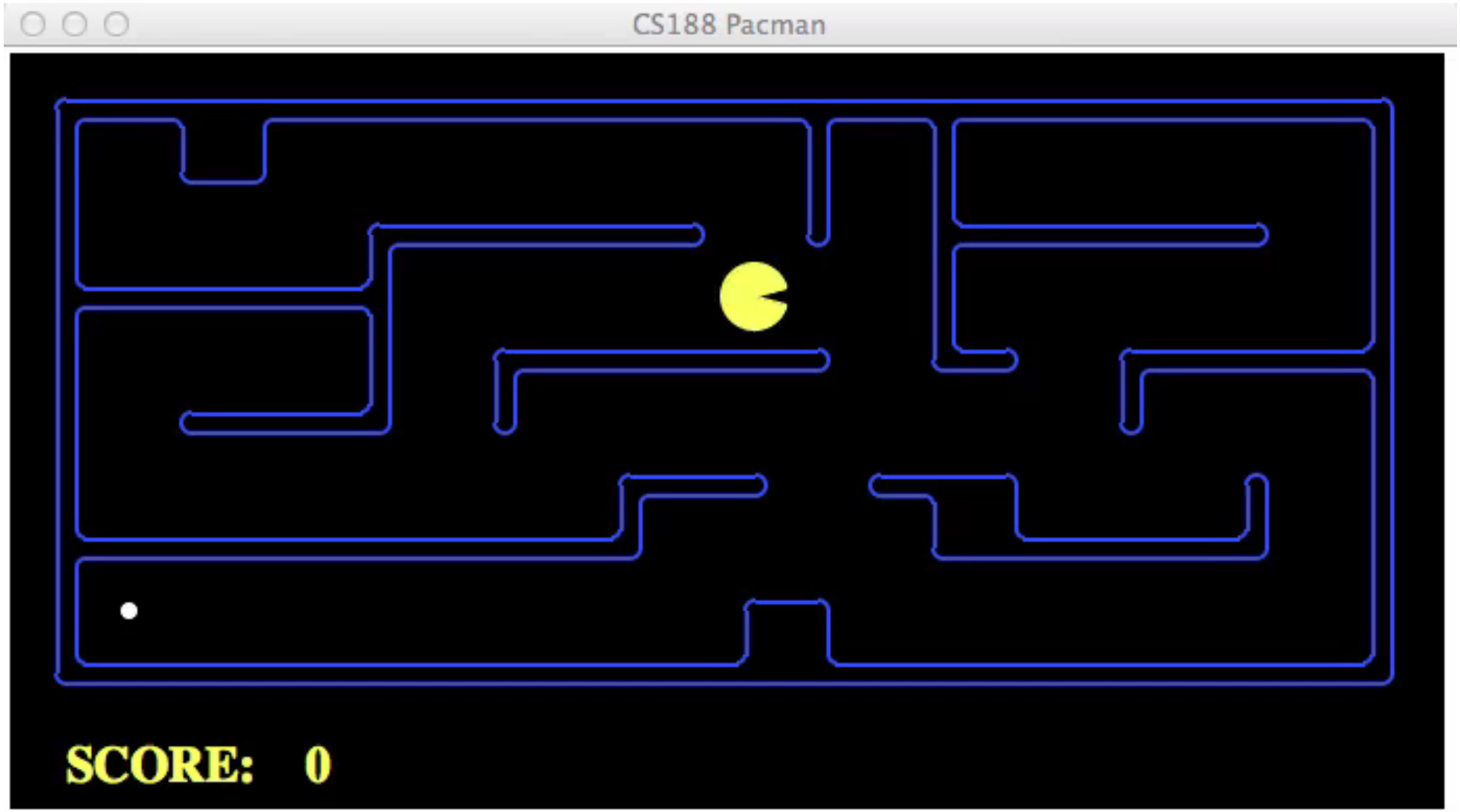
Vídeo of Demo Contours (Pacman) -- Greedy



Video of Demo Contours (Empty) – A*



Video of Demo Contours (Pacman) – A*



Comparación

Greedy

Uniform Cost

A*

Aplicaciones de A*

- Videojuegos
- Pathing / problema de rutas
- Problemas de planificación de recursos
- Planificación de
- movimientos de Robot
- Análisis de lenguaje
- Machine translation
- Speech recognition
- …

Admisibilidad, monotonía, algoritmos A*

- Sea ($e_0 \ldots e_i$) un camino del estado inicial a e_i y sea e_{i+1} un estado al que se puede acceder desde e_i. Abreviamos $g(e_0 \ldots e_i) = g(e_i)$

- Decimos que la función de coste g es **monótona** si para todo e_i, e_{i+1} se cumple que $g(e_0 \ldots e_i) = g(e_i) \leq g(e_{i+1})$

- Se dice de un heurístico h que es **admisible** si:

$$\forall x \; h(x) \leq h^*(x)$$

- Se dice de un algoritmo que es **admisible** si encuentra siempre una **solución óptima**, si esta existe. A estos algoritmos también se les llama algoritmos A*.

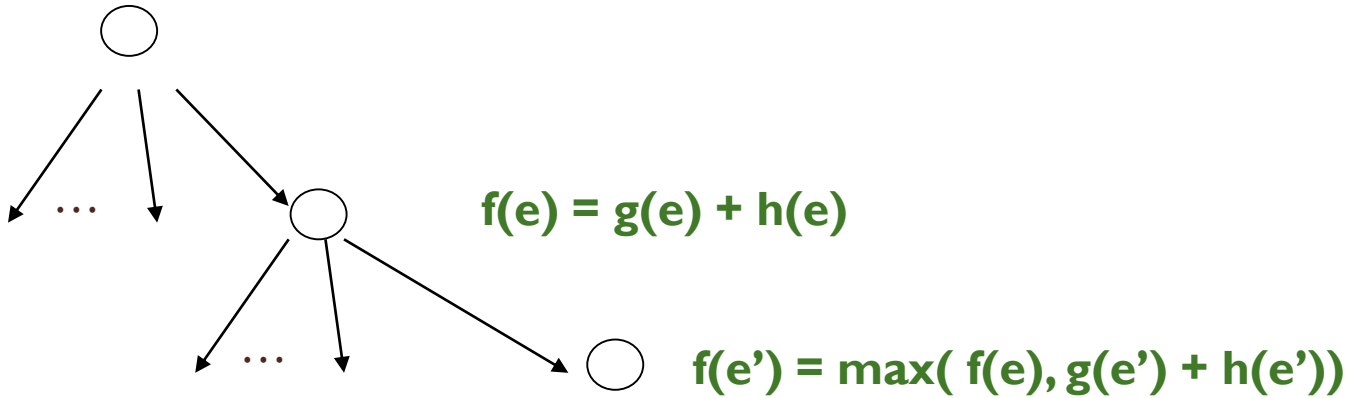
- Para asegurar la solución óptima se necesita que los heurísticos sean **admisibles**.

Propiedades de los algoritmos A*

- Sean *g* **monótona** y *h* **admisible**: ***f(e) = g(e) + h(e) = v*** nos indica que el coste de una solución que pase por ***e*** es, al menos, ***v***

- Podemos suponer que ***f*** es **monótona** si: ***f(e) ≤ f(e')***

- Si ***f* no fuera monótona**, bastaría con cambiar el algoritmo de la siguiente forma, ya que toda solución que pase por el hijo ***e'*** tiene que pasar por el padre ***e*** y su coste es, al menos, ***f(e)***.



Propiedades de los algoritmos A*

- Sea f^* el coste del camino mínimo. Para todo e en el camino, $f(e) \leq f^*$, ya que el heurístico es **admisible** → $\forall e \;\; h(e) \leq h^*(e)$

- f es **monótona** → $f(e) \leq f(e')$

- Por lo tanto, el algoritmo **A*** expande todos los nodos con $f(e) < f^*$

- La búsqueda **A*** es completa: **siempre encuentra solución**

Heurísticos más informados o dominantes

- Un heurístico ***h*** admisible es un **heurístico optimista** si, siempre estima que el coste de llegar a la solución es menor o igual que el coste real ***h****.

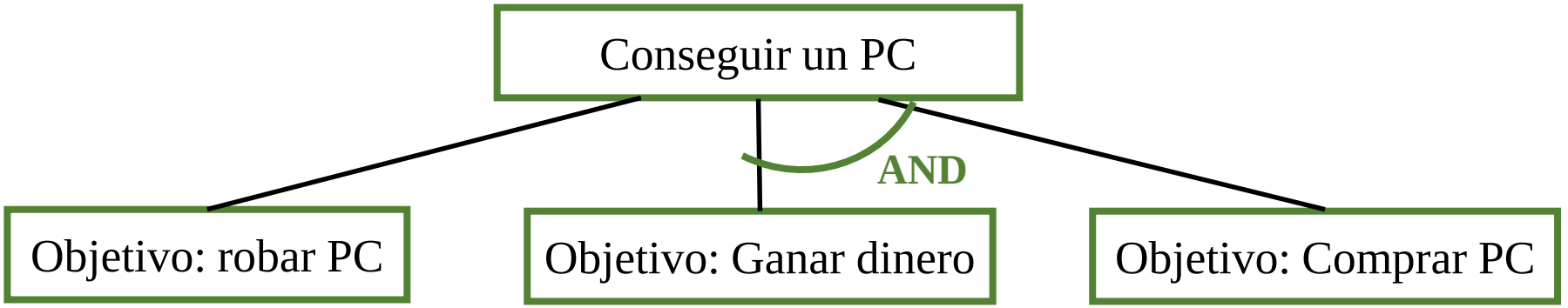
- Decimos que un heurístico h_2 está **más informado** que h_1 (o que h_2 **domina** a h_1) si $h_1 <= h_2$

- Todos los nodos ***e*** con $f(e)=g(e)+h(e)<=f^*$ se expanden en una búsqueda A*. Por lo tanto, todos los nodos ***e*** con $h(e)<=f^*- g(e)$ se expanden.

- Si h_1 y h_2 son admisibles y h_2 está más informado, siempre se cumple:
  - $h_1(e) <= h_2(e) <= h^*(e) = f^* - g(e)$. Por lo tanto, el heurístico más informado expande menos nodos y la búsqueda es más eficiente.

Grafos AND / OR

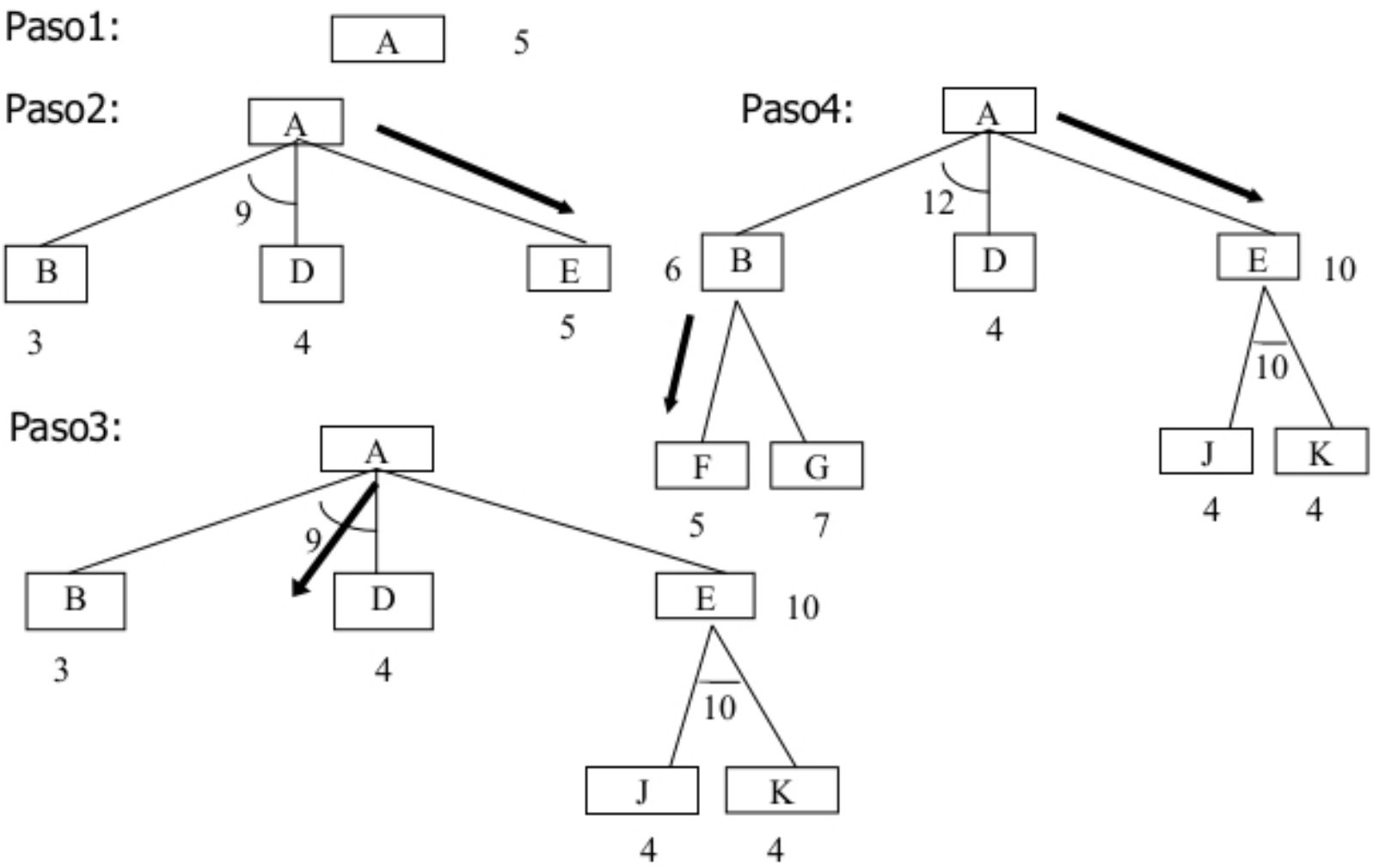
- Para problemas descomponibles
- Hasta ahora, todos los grafos eran de tipo OR: **en cada instante solo un nodo hijo**



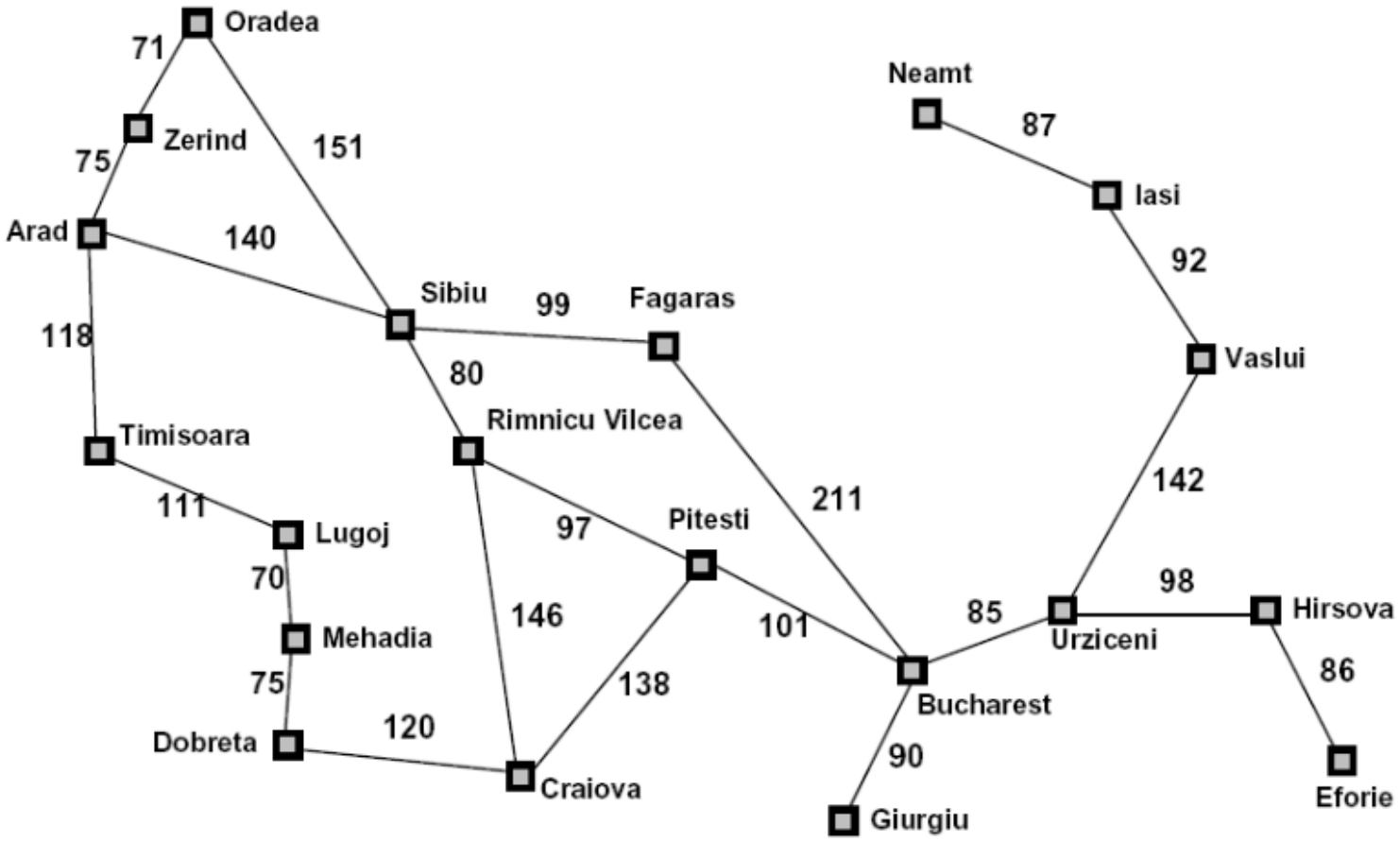
- **Características:**
  - Arcos **AND** y arcos **OR**
  - Útil para descomponer el problema en subproblemas
    - La descomposición crea arcos **AND**
  - Un arco AND puede descomponerse en cualquier nº de sucesores y todos deben resolverse para alcanzar la solución
  - Cada rama AND puede llevar a tener una solución diferente
  - Algoritmos: modificaciones del Ramificar y Acotar

Ejemplo: Grafos AND / OR

- **g=1** cada rama



Ejercicio1 A*

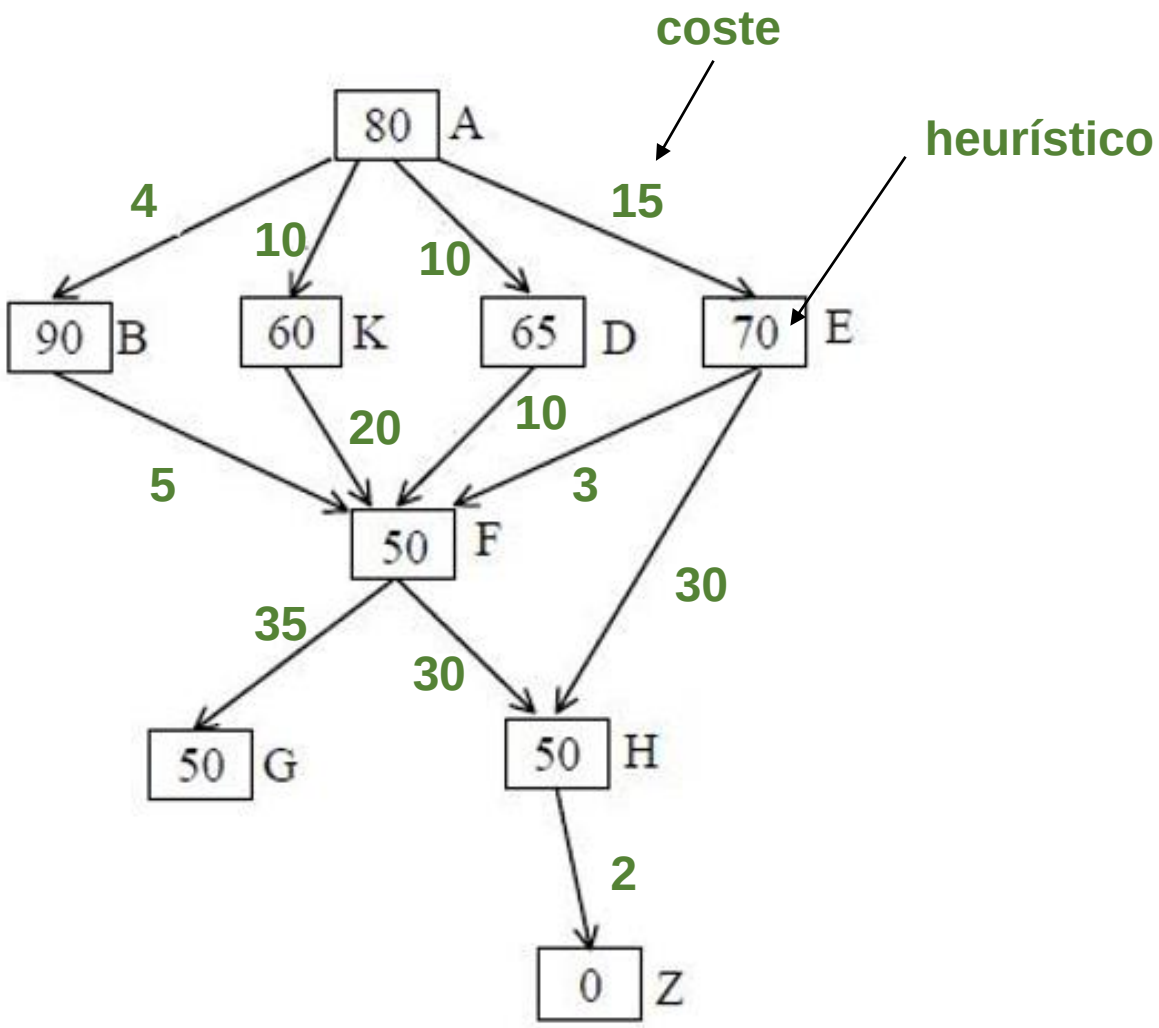


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

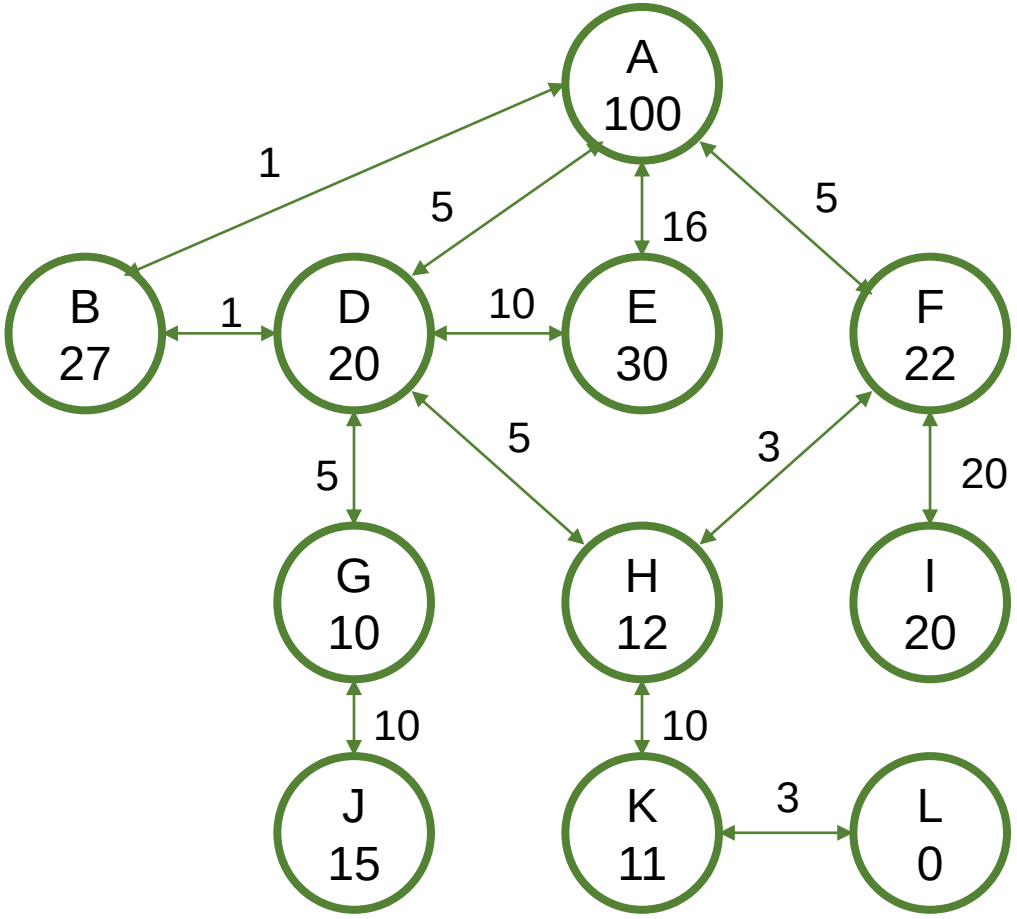
Ejercicio2 A*

- Cual es el mejor algoritmo para buscar el camino óptimo? Cual es el camino?



Ejercicio3 A*

- Aplicar Ramificar y Acotar v2 y v4. Cual es el camino al objetivo (**L**)?



- La solución es óptima?

Ejercicio4: Grafos AND / OR

- Nodo inicial N1
- Nodo final N11, N12 y N15
- Coste: 1