



# Sistemas Inteligentes

## Tema 2 - Búsqueda

*Curso 2020/2021*

Facultade de Informática  
**Universidade da Coruña**

A Coruña, 2021

- GENERALIDADES
- ESPACIO DE ESTADOS
- CARACTERÍSTICAS GENERALES DE LOS PROCESOS DE BÚSQUEDA
- ESTRATEGIAS DE EXPLORACIÓN DEL ESPACIO DE ESTADOS
- CONCLUSIONES

- El comportamiento inteligente obliga a utilizar de forma eficaz y eficiente un conjunto mínimo de conocimientos.
- Distintos problemas requieren distintas técnicas:
  - ¿Se requiere inteligencia para resolver:  $Ax^2 + Bx + C = 0$ ?
- La IA debe producir programas que:
  - Capten generalizaciones
  - Incluyan conocimiento explícito
  - Sean fácilmente actualizables
  - Puedan ser aplicados en muchas situaciones diferentes

# RESOLUCIÓN DE PROBLEMAS

- En IA el tipo de técnica está condicionado por el dominio de aplicación
- Normalmente, para resolver un problema real, no hay planteamientos exclusivos

<b>Programa de IA</b>	<b>Programa Convencional</b>
Dominios Simbólicos	Dominios Numéricos
Procesos Heurísticos	Procesos Algorítmicos
Pasos Implícitos	Pasos Explícitos
Información y Control Separados	Información y Control Integrados

- El problema de los dos cubos:
  - Disponemos de dos cubos inicialmente vacíos, uno de 8 litros, y el otro de 6 litros. Ninguno de los cubos tiene marcas ni divisiones. Disponemos también de un grifo que puede emplearse para llenar los cubos. ¿Qué tenemos que hacer para llenar el cubo de 8 litros justamente hasta la mitad?
  - NOTA: Llamaremos A al cubo de 8 litros, y B al cubo de 6 litros

## RESOLUCIÓN DE PROBLEMAS

Número de orden	Acción	Resultado
1	Llenar B	A vacío, 6 l en B
2	Vaciar B en A	6 l en A, B vacío
3	Llenar B	6 l en A, 6 l en B
4	Llenar A con B	8 l en A, 4 l en B
5	Vaciar A	A vacío, 4 l en B
6	Vaciar B en A	4 l en A, B vacío

- ¿Es la única solución?
  - Sí → Fin
  - No → Encontrar otra solución
- ¿La nueva solución encontrada es equivalente a la anterior?
- ¿Cuál es la mejor solución?
- ¿De qué depende que una solución sea mejor que otra?
- ¿Cuáles son los requisitos mínimos que debe cumplir una solución para ser aceptable?

### ■ Una representación del problema

- A = Cubo de 8 litros
- B = Cubo de 6 litros
- [A] = Contenido de A
- [B] = Contenido de B
- ([A],[B]) = Estado actual del problema
- Número “n” = Acción ejecutada



## RESOLUCIÓN DE PROBLEMAS

Número de orden	Precondiciones	Acción
1	A no lleno	Llenar A
2	B no lleno	Llenar B
3	A no vacío	Vaciar A
4	B no vacío	Vaciar B
5	A no vacío, y B no lleno, y $[A] + [B] \leq 6$	Vaciar A en B
6	B no vacío, y A no lleno, y $[A] + [B] \leq 8$	Vaciar B en A
7	B no vacío, y A no lleno, y $[A] + [B] \geq 8$	Llenar A con B
8	A no vacío, y B no lleno, y $[A] + [B] \geq 6$	Llenar B con A

- Secuencia de acciones que nos lleva a una solución del problema:

**(0,0) 2 (0,6) 6 (6,0) 2 (6,6) 7 (8,4) 3 (0,4) 6 (4,0)**

- Espacio de Estados (EE) = Descripción formal del universo de discurso
  - Conjunto de estados iniciales
  - Conjunto de operadores que definen operaciones permitidas entre estados
  - Conjunto de estados meta u objetivos: soluciones aceptables, aunque no necesariamente la mejor

- La resolución de un problema en IA consiste en:
  1. La aplicación de un conjunto de técnicas conocidas, cada una de ellas definida como un paso simple en el espacio de estados
  2. Un proceso de búsqueda, o estrategia general de exploración del espacio de estados

Para abordar un problema desde la óptica de la inteligencia artificial tenemos que ser capaces de construir un modelo computacional del universo de discurso, o dominio del problema

- FORMALMENTE...
  - **I es el conjunto de estados iniciales...**  
 $I = \{i1, i2, \dots, in\}$
  - **O es el conjunto de operaciones permitidas...**  
 $O = \{o1, o2, \dots, om\}$
  - **M es el conjunto de metas o soluciones aceptables...**  
 $M = \{m1, m2, \dots, mt\}$
  - **Búsqueda = proceso de exploración en el espacio de estados tal que...**  
 $O : ( I \rightarrow M )$
  - **Paso Simple...  $ox : ( iz \rightarrow iw )$  con  $iz, iw \in I, ox \in O$**
  - **Si  $iw \in M$ , entonces  $iw$  verifica la prueba de meta, o test de realización, y es una solución del problema**

- El espacio de estados es una representación exclusivamente del dominio del problema, no indica cómo obtener la solución
- Necesitamos también procesos de búsqueda o mecanismos generales de exploración del espacio de estados
  - Criterios de selección y aplicación de operadores relevantes
  - Capacidad de decisión sobre cuál será el próximo movimiento
  - Búsqueda sistemática
  - Tratar de generar siempre estados nuevos, no generados previamente

# RESOLUCIÓN DE PROBLEMAS

- Volviendo al problema de los dos cubos...
  - $I = (0,0)$
  - $M = (4,x) / x \in [0,6]$
  - $O = \{op1, op2, op3, op4, op5, op6, op7, op8\}$
  
- **Estrategia 1**
  1. *Identificar estado inicial*
  2. *Prueba de meta*
    - *Si meta*  $\rightarrow$  *Fin*
    - *Si no meta*  $\rightarrow$  *Seguir (ir a 3)*
  3. *Aplicar 1er operador que cumpla requisitos de estado actual, según su número de orden*
  4. *Prueba de meta*
    - *Si meta*  $\rightarrow$  *Fin*
    - *Si no meta*  $\rightarrow$  *Seguir (ir a 3)*
  
- Secuencia de estados generada: **(0,0)1(8,0)2(8,6)3(0,6)1(8,6)...**
- Hemos caído en un bucle
- La estrategia es sistemática, pero no siempre genera estados nuevos, y hay operadores que se repiten

## -Estrategia 2

1. *Identificar estado inicial*
2. *Prueba de meta*
  - *Si meta* → *Fin*
  - *Si no meta* → *Ir a 3*
3. *Seleccionar operadores que verifiquen precondiciones*
4. *Descartar los ya aplicados*
5. *Aplicar el primero de los supervivientes*
6. *Prueba de meta*
  - *Si meta* → *Fin*
  - *Si no meta* → *Ir a 3*

-Secuencia de estados: **(0,0)1(8,0)2(8,6)3(0,6)4(0,0)**

-Búsqueda detenida sin solución

op1 a op4 ya aplicados, op5 a op8 no verifican restricciones

-Necesidad de utilizar estructuras adicionales (autoconocimiento)

## -Estrategia 3

1. *Identificar estado inicial*
2. *Prueba de meta*
  - *Si meta* → *Fin*
  - *Si no meta* → *Ir a 3*
3. *Seleccionar operadores que verifiquen precondiciones*
4. *Descartar los ya aplicados*
5. *Descartar operadores que no generen estados nuevos*
6. *Aplicar el primero de los supervivientes*
7. *Prueba de meta*
  - *Si meta* → *Fin*
  - *Si no meta* → *Ir a 3*

-Secuencia de estados: **(0,0)1(8,0)2(8,6)3(0,6)6(6,0)**

-Búsqueda detenida sin solución

op1, op2, op3, op6 ya se han aplicado; op4, op7 y op8 no verifican restricciones, op5 genera estado repetido.

-Más estructuras adicionales (incrementar el autoconocimiento)



## -Estrategia 4

1. *Identificar estado inicial*
2. *Prueba de meta*
  - *Si meta* → *Fin*
  - *Si no meta* → *Ir a 3*
3. *Seleccionar operadores que verifiquen precondiciones*
4. *Descartar operadores que no generen estados nuevos*
5. *Aplicar el primero de los supervivientes*
6. *Prueba de meta*
  - *Si meta* → *Fin*
  - *Si no meta* → *Ir a 3*

-Secuencia de estados...

**(0,0)1(8,0)2(8,6)3(0,6)6(6,0)2(6,6)7(8,4)3(0,4)6(4,0)**

-Solución aceptable, aunque no óptima

-Más estructuras adicionales (incrementar el autoconocimiento)

# RESOLUCIÓN DE PROBLEMAS

## Cuestiones importantes:

- Hemos definido de forma independiente el conocimiento (operadores y precondiciones) de la forma (estrategias) en que dicho conocimiento es empleado.
  - Estrategias de control de conocimiento reutilizables.
- Hay una separación clara entre el conocimiento del sistema y las estrategias de control del conocimiento
- Son soluciones aceptables aquéllas que cumplen las restricciones de la prueba de meta (¿qué pasaría si del grifo manara Hg?)
- El sistema necesita AUTOCONOCIMIENTO (en forma de estructuras adicionales auxiliares), para funcionar correctamente

## CARACTERÍSTICAS GENERALES DE LOS PROCESOS DE BÚSQUEDA

1. DIRECCIÓN DE LA BÚSQUEDA
2. TOPOLOGÍA DEL PROCESO
3. REPRESENTACIÓN DE LOS ESTADOS POR LOS QUE DISCURRE LA SOLUCIÓN DEL PROBLEMA
4. CRITERIOS DE SELECCIÓN Y APLICACIÓN DE OPERADORES RELEVANTES
5. OPTIMIZACIÓN DE LA BÚSQUEDA MEDIANTE EL USO DE FUNCIONES HEURÍSTICAS

- DIRECCIÓN DEL PROCESO DE BÚSQUEDA
  - Desde los estados iniciales hacia los estados meta
    - Proceso progresivo o dirigido por los datos
  - Desde los estados meta hacia los estados iniciales
    - Proceso regresivo, evocativo, o dirigido por objetivos

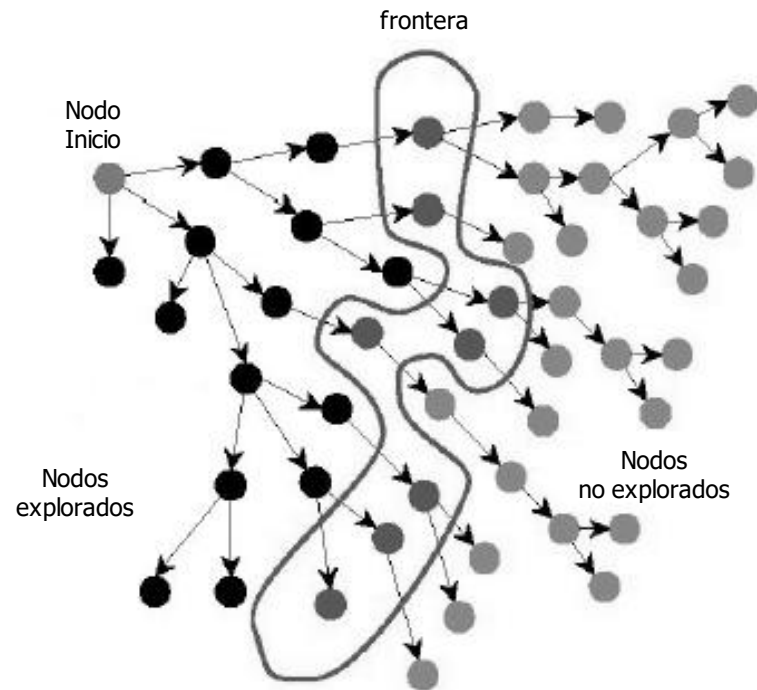
¿Cuándo, y en función de qué, es mejor una dirección que otra?

1. Tamaño relativo de los conjuntos I y M del espacio de estados
  2. Factor de ramificación, o número promedio de estados que podemos alcanzar directamente desde un estado dado
  3. Inclusión de estructuras explicativas como requisito inicial en el diseño de nuestro sistema inteligente
- 
1. De menor a mayor número de estados
  2. En el sentido del menor factor de ramificación
  3. De la forma que mejor se ajuste al modo de razonar de expertos y usuarios

- TOPOLOGÍA DEL PROCESO DE BÚSQUEDA
  - ÁRBOL
  - GRAFO

# Topología del proceso de búsqueda

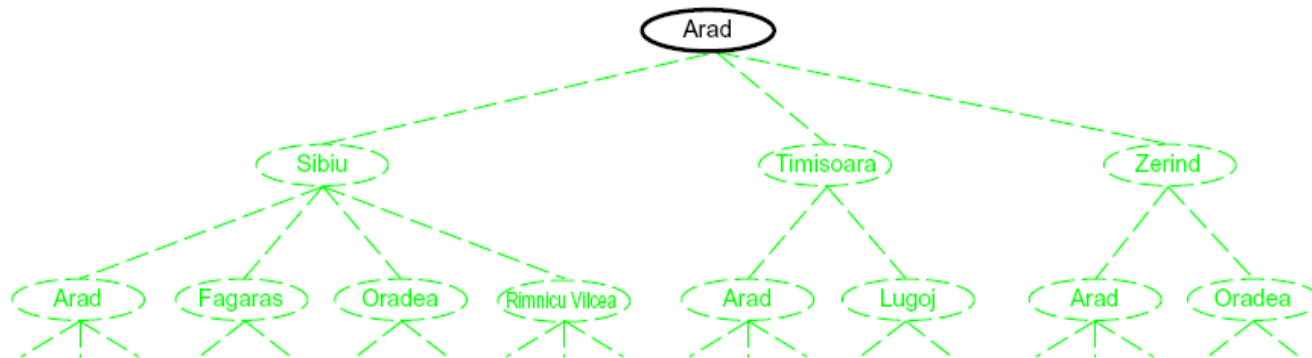
- Construcción de un árbol de búsqueda superpuesto al espacio de estados
  - Raíz (estado inicial), hojas (estados sin sucesor)
  - En cada paso seleccionar una hoja y expandir



# Búsqueda de soluciones

## Introducción

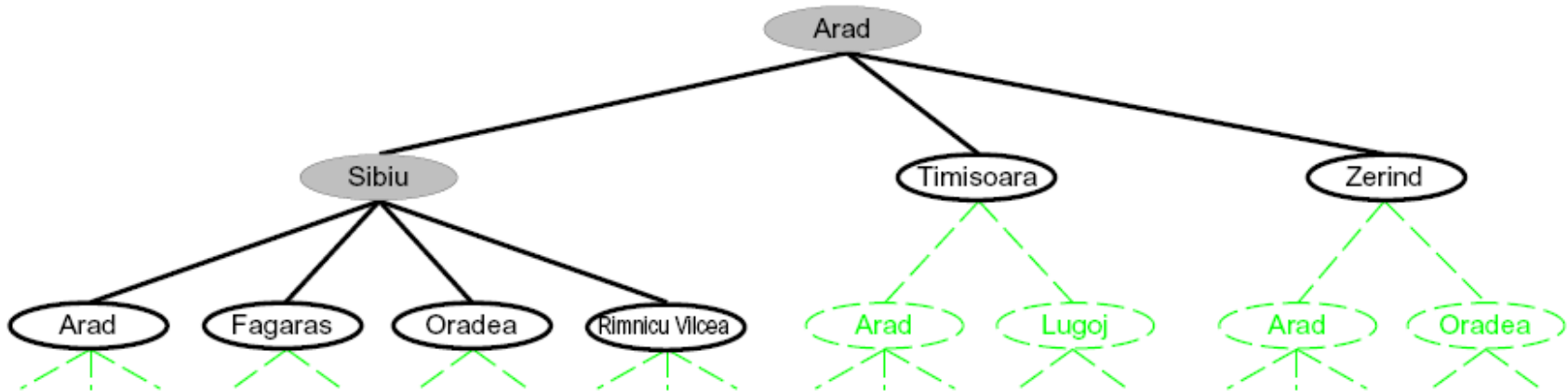
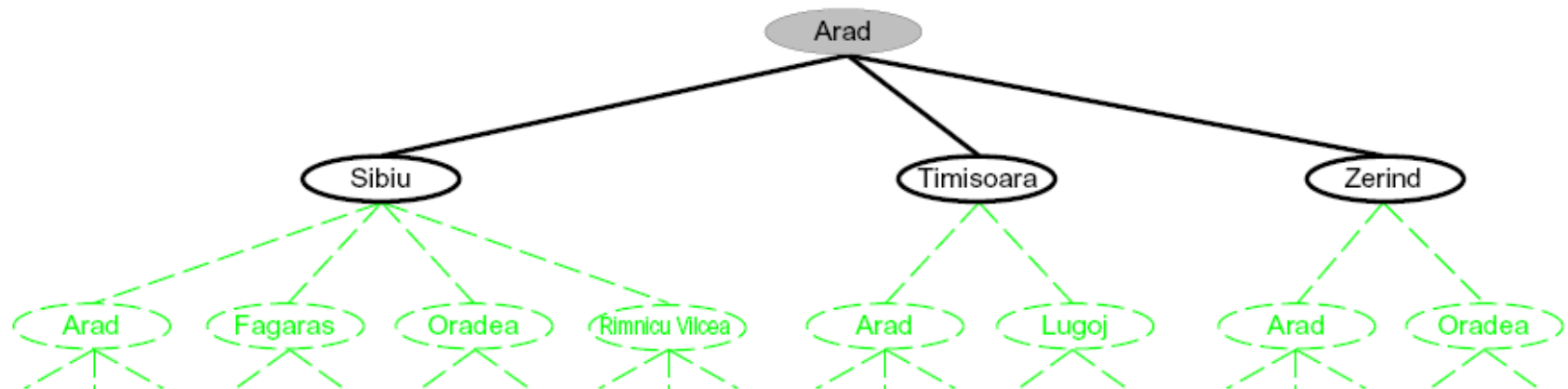
- Los algoritmos de búsqueda construyen secuencias de acciones a partir de un estado inicial. Estas secuencias forman un árbol de búsqueda:
  - Raíz: estado inicial
  - Ramas: acciones
  - Nodos: estados del espacio de estados
- Ejemplo: Encontrar una ruta desde Arad a Bucarest





# Búsqueda de soluciones

## Introducción



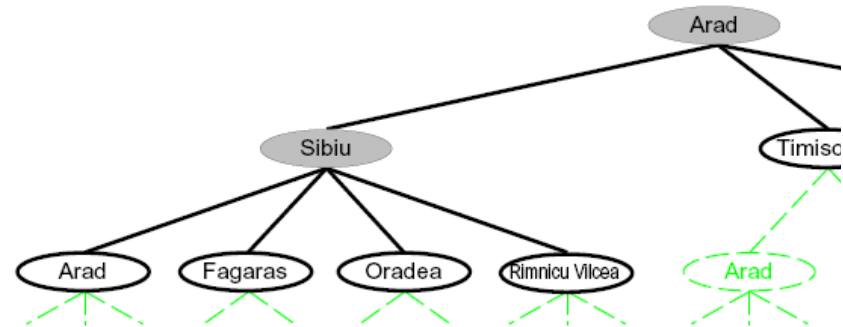
```
función BUSQUEDA-ARBOL (problema) produce solución, o fallo
  inicializar la frontera usando el estado inicial del problema
  bucle hacer
    si la frontera esta vacía entonces devolver fallo
    sino escoger un nodo hoja y eliminarlo de la frontera
    si el nodo hoja contiene un estado meta, entonces devolver la solución
    sino expandir el nodo elegido y añadir los nodos resultantes a la frontera
  fin
```

Descripción informal del algoritmo general de búsqueda en árboles

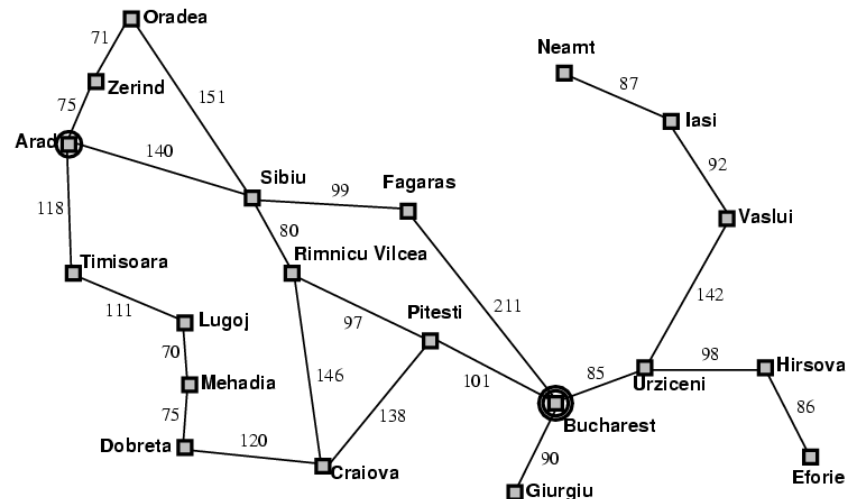
# Búsqueda de soluciones

## Estados repetidos

- Bucles: Arad y Sibiu



- Caminos redundantes



## Topología de búsqueda: Árbol vs Grafo

**función** BUSQUEDA-ARBOL (*problema*) **produce** solución, o fallo  
inicializar la frontera usando el estado inicial del *problema*  
**bucle hacer**  
    **si** la frontera esta vacía **entonces devolver** fallo  
    **sino** escoger un nodo hoja y eliminarlo de la frontera  
    **si** el nodo hoja contiene un estado meta, **entonces** devolver la solución  
    **sino** expandir el nodo elegido y añadir los nodos resultantes a la frontera  
**fin**

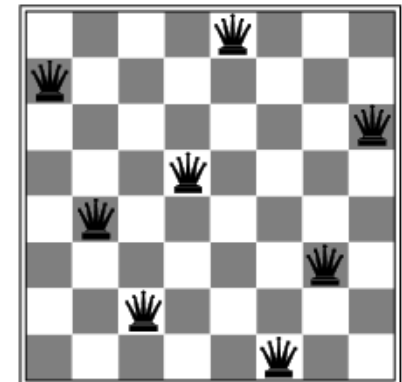
**función** BUSQUEDA-GRAFO (*problema*) **produce** solución, o fallo  
inicializar la frontera usando el estado inicial del *problema*  
*inicializar el conjunto de explorados como vacío*  
**bucle hacer**  
    **si** la frontera esta vacía **entonces devolver** fallo  
    **sino** escoger un nodo hoja y eliminarlo de la frontera  
    **si** el nodo hoja contiene un estado meta, **entonces** devolver la solución  
    **sino añadir el nodo al conjunto de explorados**  
        expandir el nodo elegido y añadir los nodos resultantes a la frontera  
        solo si no están en la frontera o conjunto de explorados  
**fin**

Descripción informal del algoritmo general de búsqueda en grafos

# Topología del proceso de búsqueda

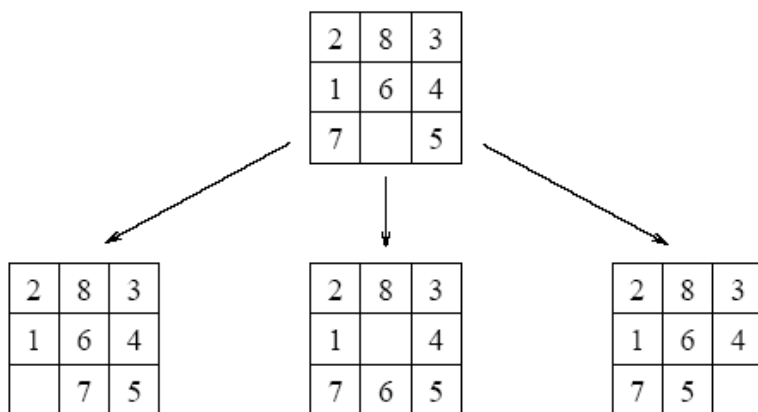
- El espacio de estados (finito) no es igual al árbol de búsqueda (¿infinito?)
- ¿Árbol o Grafo? La formulación eficiente del problema evita repetir estados.
- Meta. 8 reinas en tablero sin atacarse entre sí
  - Opción 1: Formulación incremental
    - Estados: cualquier combinación de 0 a 8 reinas en el tablero
    - Estado inicial: ninguna reina
    - Acciones: Añadir una reina en una casilla libre
    - Modelo de transición: tablero con reina añadida en la casilla específica
    - Nº combinaciones a investigar =  $64 * 63 * \dots * 57 \approx 1,8 * 10^{14}$
  - Opción 2: Formulación de estado completo
    - Estado: 0 a 8 reinas, una por columna desde la columna más a la izquierda, sin atacarse
    - Acciones: Añadir una reina en cualquier casilla en la columna más a la izquierda vacía, tal que no se vea atacada
    - Nº combinaciones a investigar -> 2057. Cada estado se alcanza por un único camino.

	Árbol	Grafo
Memoria	↑	↓
Eficiencia	↑	↓

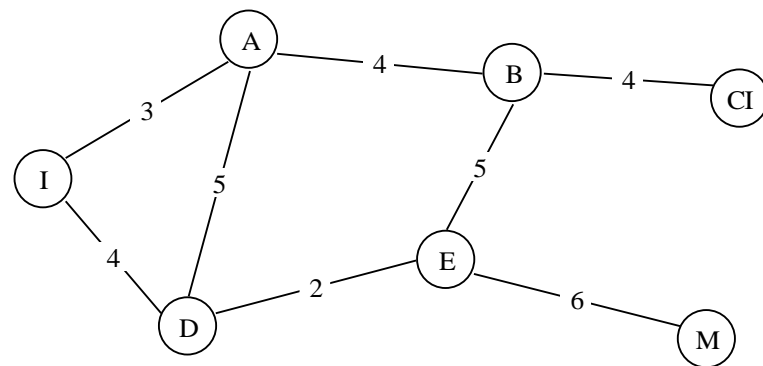


# Topología del proceso de búsqueda

- Cuando existen operadores reversibles, la repetición es inevitable



Problema del 8 puzzle



Problema del viajante

# Topología del proceso de búsqueda

- Transformación de árbol en grafo
  1. Empezar generando estado(s) tras aplicar operador(es) relevante(s)
  2. Para cada nuevo estado generado
    - Si es nuevo añadirlo y volver a (1)
    - Si no es nuevo descartarlo e ir a (3)
  3. Añadir un nuevo enlace entre el nodo que se está expandiendo y el sucesor
  4. Recorrer el nuevo camino desde el principio
    - Si es más corto insertarlo como mejor camino, propagar el cambio, reorganizar el grafo y volver a (1)
    - Si no es más corto volver a (1)

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set
```

**Figure 3.7** An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

- La utilización de grafos de búsqueda reduce los esfuerzos de exploración en el espacio de estados
- La utilización de grafos de búsqueda obliga a comprobar si cada estado generado ha sido generado ya en pasos anteriores
- La utilización de grafos de búsqueda demanda menos recursos de memoria, pero suponen un coste computacional mayor
- Los árboles de búsqueda son más eficientes desde una perspectiva computacional, pero tienen mayores problemas de memoria



## RESOLUCIÓN DE PROBLEMAS: El problema de la representación

1. Representación de objetos, entidades relevantes, o hechos del dominio (NODOS)
    - Representación estática del dominio
  2. Representación de relaciones entre objetos, entidades relevantes, o hechos del dominio (OPERADORES, ACCIONES)
    - Representación dinámica del dominio
  3. Representación de las secuencias de estados surgidas durante los procesos de búsqueda (ESTRATEGIAS)
    - Representación dinámica del proceso de búsqueda de soluciones
- Las tres representaciones están estrechamente relacionadas

### SELECCIÓN SISTEMÁTICA DE OPERADORES RELEVANTES

- Al proceso de selección sistemática de operadores relevantes se denomina **emparejamiento**.
  - Reconocer operadores aplicables al estado actual
  - Es una de las tareas más costosas de los programas de IA
  - Dos tipos:
    - literal
    - con variables

- **Emparejamiento literal**

- Búsqueda simple de operadores del conjunto  $O$  del espacio de estados
  - A través de las precondiciones del operador si el proceso es dirigido por los datos
  - A través de los consecuentes del operador si el proceso es dirigido por los objetivos
- Problemas:
  - Si  $O$  es muy grande el emparejamiento literal es muy ineficiente
  - No es siempre evidente qué operador es aplicable

- Casos particulares del emparejamiento literal
  - El operador empareja completamente con el estado
  - Las precondiciones del operador son un subconjunto de la descripción del estado actual
  - Las precondiciones del operador coinciden sólo parcialmente con la descripción del estado actual
  - Las precondiciones del operador no coinciden con la descripción del estado actual
- Útil en dominios pequeños

## ■ Emparejamiento con variables

- De naturaleza no literal
- Útil cuando el problema requiere una búsqueda extensa en la que haya variables involucradas

## ■ EJEMPLO:

- Hechos de un dominio:

– Juan es hijo de María	→	HIJO(María,Juan)
– Pedro es hijo de Juan	→	HIJO(Juan,Pedro)
– Tomás es hijo de Pedro	→	HIJO(Pedro,Tomás)
– Rosa es hija de Pedro	→	HIJA(Pedro,Rosa)
– Ana es hija de Juan	→	HIJA(Juan,Ana)
– Rosa es hija de Ana	→	HIJA(Ana,Rosa)

- Operadores del dominio:

– Op1: HIJO(x,y) and HIJO(y,z)	→	NIETO(x,z)
– Op2: HIJA(x,y) and HIJO(y,z)	→	NIETO(x,z)
– Op3: HIJO(x,y) and HIJA(y,z)	→	NIETA(x,z)

# Emparejamiento con variables

## Hechos del dominio

Juan es hijo de María	HIJO (María, Juan)
Pedro es hijo de Juan	HIJO (Juan, Pedro)
Tomás es hijo de Pedro	HIJO (Pedro, Tomás)
Rosa es hija de Pedro	HIJA (Pedro, Rosa)
Ana es hija de Juan	HIJA (Juan, Ana)
Rosa es hija de Ana	HIJA (Ana, Rosa)

## Operadores del dominio

op1:  $\text{HIJO}(x,y) \text{ and } \text{HIJO}(y,z) \rightarrow \text{NIETO}(x,z)$

op2:  $\text{HIJA}(x,y) \text{ and } \text{HIJO}(y,z) \rightarrow \text{NIETO}(x,z)$

op3:  $\text{HIJO}(x,y) \text{ and } \text{HIJA}(y,z) \rightarrow \text{NIETA}(x,z)$

## Problema

Dados los hechos anteriores (estado inicial) encontrar un estado meta que incluya los mismos hechos y un hecho nuevo que indique quién es el nieto de Juan.

# Emparejamiento con variables

## Hechos del dominio

Juan es hijo de María	HIJO (María, Juan)
Pedro es hijo de Juan	HIJO (Juan, Pedro)
Tomás es hijo de Pedro	HIJO (Pedro, Tomás)
Rosa es hija de Pedro	HIJA (Pedro, Rosa)
Ana es hija de Juan	HIJA (Juan, Ana)
Rosa es hija de Ana	HIJA (Ana, Rosa)

## Operadores del dominio

- op1:  $HIJO(x,y) \text{ and } HIJO(y,z) \rightarrow NIETO(x,z)$
- op2:  $HIJA(x,y) \text{ and } HIJO(y,z) \rightarrow NIETO(x,z)$
- op3:  $HIJO(x,y) \text{ and } HIJA(y,z) \rightarrow NIETA(x,z)$

## Solución

- Interesa usar op1-op2, porque son los operadores que concluyen sobre la existencia de algún nieto ( $x=Juan$ )
- Hay que encontrar un “y” que verifique, para algún valor de “z”, que

*$HIJO(Juan,y) \text{ and } HIJO(y,z)$*

# Emparejamiento con variables

## Hechos del dominio

Juan es hijo de María	HIJO (María, Juan)
Pedro es hijo de Juan	HIJO (Juan, Pedro)
Tomás es hijo de Pedro	HIJO (Pedro, Tomás)
Rosa es hija de Pedro	HIJA (Pedro, Rosa)
Ana es hija de Juan	HIJA (Juan, Ana)
Rosa es hija de Ana	HIJA (Ana, Rosa)

## Operadores del dominio

- op1:  $\text{HIJO}(x,y) \text{ and } \text{HIJO}(y,z) \rightarrow \text{NIETO}(x,z)$
- op2:  $\text{HIJA}(x,y) \text{ and } \text{HIJO}(y,z) \rightarrow \text{NIETO}(x,z)$
- op3:  $\text{HIJO}(x,y) \text{ and } \text{HIJA}(y,z) \rightarrow \text{NIETA}(x,z)$

## Solución

- Opción 1: Comprobar a todos los hijos de Juan y verificar que alguno de ellos tiene un hijo
- Opción 2: Comprobar que de todos los que tienen algún hijo, hay alguno que es hijo de Juan



## ■ Resolución de Conflictos

- Relacionado con el emparejamiento, un conflicto es una situación en la que más de un operador empareja con nuestro estado actual
- En función de la estrategia, cuando aparece un conflicto... ¿Qué operador(es) ejecutamos?

## ■ Reglas de carácter general

1. Si podemos evitarlo no aplicaremos operadores ya aplicados
2. Trataremos de aplicar primero operadores que emparejen con hechos recientemente incorporados
3. Aplicaremos primero operadores con precondiciones más restrictivas
4. De no poder discriminar con las pautas anteriores, ejecutaremos uno (o más, depende de la estrategia), al azar

## Funciones Heurísticas

- Funciones de carácter numérico que nos permiten estimar el beneficio de una determinada transición en el espacio de estados
- Se utilizan para optimizar los procesos de búsqueda
- Tratan de guiar la exploración del espacio de estados de la forma más provechosa posible
- Sugieren el mejor camino a priori, cuando disponemos de varias alternativas
- El estudio de las funciones heurísticas se denomina Heurética

- Estrategias de exploración del espacio de estados (búsqueda)
  - La eficacia determinada por la estrategia empleada, y por los mecanismos diseñados para controlar la aplicación del conocimiento del dominio
- Técnicas de búsqueda de propósito específico
  - Diseñadas “ad hoc” para resolver un problema concreto
- Técnicas de búsqueda de propósito general
  - Métodos débiles de exploración del espacio de estados
  - Cualquier método débil de exploración configura una búsqueda que será:
    - En anchura
    - En profundidad
    - Mixta anchura-profundidad

# Estrategias de búsqueda: Evaluación

- Una estrategia se define por el orden de expansión de los nodos.
- Se evalúan de acuerdo a:
  - Completitud:
    - ¿Encuentra siempre la solución? (si existe)
  - Complejidad temporal:
    - ¿Cuánto tarda en encontrar una solución?
  - Complejidad espacial:
    - ¿Cuánta memoria necesita?
  - Optimización:
    - ¿encuentra siempre la solución de mayor calidad? (si existen varias soluciones)

# Estrategias de búsqueda: Evaluación

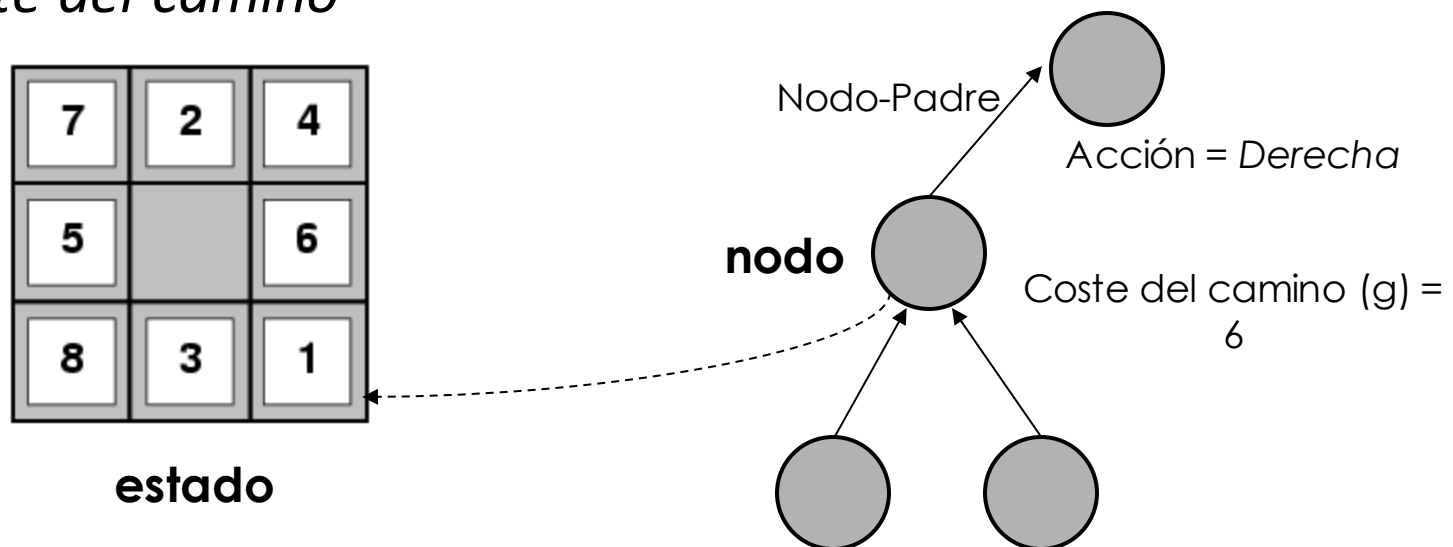
- La complejidad temporal y espacial se miden en términos de:
  - $b$ : factor de ramificación (número promedio de sucesores por nodo)
  - $d$ : profundidad de la solución menos costosa (o del nodo meta óptimo)
  - $m$ : máxima profundidad del espacio de estados (o de cualquier camino)
  - Complejidad Temporal  $\rightarrow$  máx. número nodos generados
  - Complejidad Espacial  $\rightarrow$  máx. número nodos almacenados
- Efectividad:

Coste total = coste búsqueda + coste camino

# Búsqueda de soluciones

## Estructuras de datos

- Espacio de estados  $\neq$  árbol de búsqueda
- Estado, corresponde a una configuración del mundo
- Nodo, estructura de datos contable que representa el árbol de búsqueda
- Dos nodos diferentes pueden contener el mismo estado
- Componentes de un nodo: *estado*, *nodo padre*, *acción* y *coste del camino*



- **Formulación de metas, formulación del problema, búsqueda y solución** son los pasos básicos de la solución de problemas
- Los componentes formales de un problema de búsqueda son **estado inicial, acciones disponibles, modelo de transición, prueba de meta y función de coste** del camino. Los tres primeros definen implícitamente el **espacio de estados** del problema
- Una **solución** es una secuencia de acciones que transforma el estado inicial en el estado meta. La **óptima** es la que tiene el menor coste de camino
- Los **algoritmos de búsqueda** se distinguen por el estado que seleccionan para expansión, lo cual se llama también **estrategia de búsqueda**

## Lo que no debes olvidar (V)

- Los algoritmos de búsqueda trabajan considerando varias posibles secuencias de acciones que parten del estado inicial y forman un **árbol de búsqueda** cuya raíz es el estado inicial, las ramas son las acciones y los nodos corresponden a estados del espacio de estados del problema.
- La **búsqueda basada en grafo** evita el problema de explorar caminos redundantes. Para ello es necesario una estructura que recuerde cada nodo previo explorado.
- Para evaluar el rendimiento de los algoritmos de búsqueda exploramos las propiedades de **completitud, optimización, complejidad en tiempo y en espacio**.



# Estrategias de búsqueda: Implementación

- El espacio de estados define un **grafo** de búsqueda implícito (nodo inicial y función sucesor), no un árbol. Los algoritmos sólo conservarán un padre para cada nodo
- **Expansión** de un nodo: aplicación de operadores relevantes y generación de sucesores
- Listas de *abiertos* (**frontera**): nodos en espera de ser expandidos
  - Separa el grafo espacio de estados en la región explorada y la región sin explorar
  - Cualquier camino desde el estado inicial a un estado sin explorar tiene que pasar a través de un estado en la frontera
- Lista de *cerrados* (**explorados**): nodos expandidos. A efectos de implementación son colas.
- **Estrategia** de búsqueda: selección del siguiente nodo a expandir de la lista de abiertos. El tipo de cola afectará al orden de la búsqueda.
- Tratamiento de nodos repetidos: No se expande el nodo actual si pertenece a la lista de cerrados.

# Estrategias de búsqueda: Implementación

```
función BUSQUEDA–GRAFO (problema, abiertos)
devuelve una solución o fallo
    cerrados ← un conjunto vacío
    abiertos ← INSERTAR (HACER_NODO (
                        ESTADO_INICIAL[problema]), abiertos)

    bucle hacer
        /* No hay candidatos para la expansion */
        if ESVACIA (abiertos) entonces devolver fallo

        /* Elegir un nodo hoja para expansion */
        nodo ← QUITAR_FRENTE (abiertos)

        si TEST_META[problema] (ESTADO[nodo])
            entonces devuelve SOLUCIÓN(nodo)
        si ESTADO[nodo] no esta en cerrados entonces
            añadir ESTADO[nodo] a cerrados
            abiertos ← INSERTAR–TODOS (
                        EXPANDIR(nodo,problema), abiertos)
```

# Estrategias de búsqueda: Implementación

```
funcion EXPANDIR (nodo,problema)
devuelve un conjunto de nodos

    sucesores  $\leftarrow$  el conjunto vacio
    para cada par <accion,resultado> en
        FUNCION_SUCESOR[problema] aplicada a ESTADO[nodo] hacer
            s  $\leftarrow$  un nuevo NODO
            ESTADO[s]  $\leftarrow$  resultado
            NODO_PADRE[s]  $\leftarrow$  nodo
            ACCION[s]  $\leftarrow$  accion
            COSTE_CAMINO[s]  $\leftarrow$  COSTE_CAMINO[nodo] +
                                COSTE_PASO (nodo,accion,s)
            PROFUNDIDAD[s]  $\leftarrow$  PROFUNDIDAD[nodo] + 1
            anadir s a sucesores
    devolver sucesores
```

- La estructura de datos apropiada para la frontera es una cola
- Operaciones en la cola:
  - EMPTY? (*queue*): devuelve TRUE si no hay más elementos en la cola
  - POP (*queue*) elimina el primer elemento de la cola y lo devuelve
  - INSERT (*element*, *queue*) Inserta un elemento y devuelve la cola resultante.
- Variantes del orden de almacenamiento:
  - FIFO: pop del elemento más antiguo de la cola
  - LIFO (stack): pop del elemento más reciente de la cola
  - COLA DE PRIORIDAD: pop del elemento de la cola de mayor prioridad
- Estructura de datos para explorados: tabla hash

- *No informada o ciega*: no disponen de información adicional acerca de los estados más allá de la que proporciona la definición del problema
- *Informada o heurística*: disponen de conocimiento específico del problema para alcanzar el objetivo de manera más eficiente
  - *Heurística* (del griego “*heurisko*”, “*yo encuentro*”): conocimiento parcial sobre un problema/dominio que permite resolver problemas eficientemente en ese problema/dominio



# Estrategias de búsqueda no informada (búsqueda a ciegas)

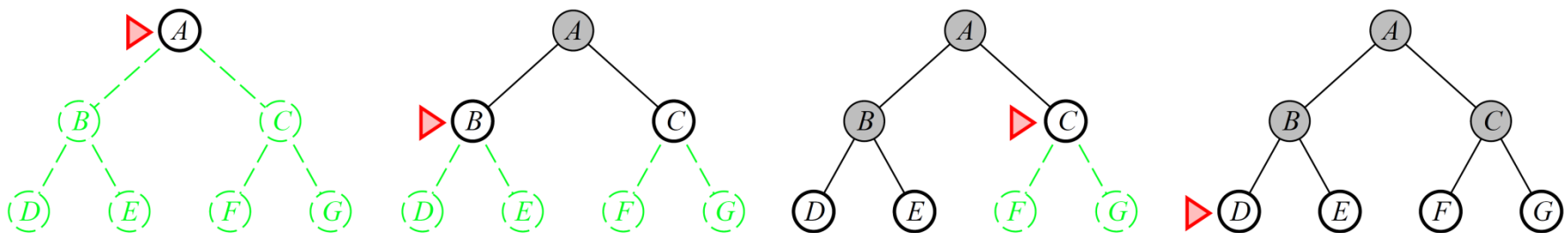


# Estrategias no informadas

- Las búsquedas ciegas no tienen preferencia sobre el siguiente estado (nodo) a expandir.
- Los distintos tipos de búsquedas ciegas se caracterizan por el orden de expansión de los nodos.
- Esto tendrá un efecto directo sobre la calidad del proceso con respecto a los cuatro criterios definidos.
- Tipos:
  - Preferente por amplitud: Variante de coste uniforme
  - Preferente por profundidad: Variante limitada en profundidad y profundización iterativa

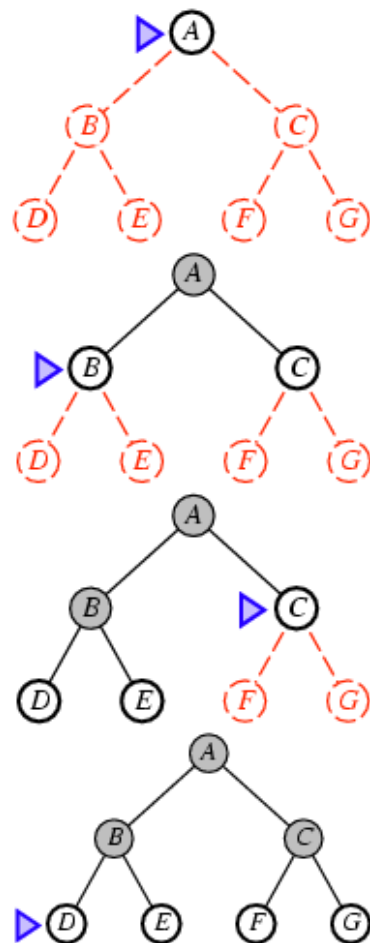
## Búsqueda preferente en amplitud (*breadth-first*)

- Expandir primero el nodo raíz, luego los sucesores de nodo raíz, luego todos los sucesores de los sucesores,...
- Se expanden todos los nodos de un mismo nivel antes de expandir nodos de niveles inferiores
- Particularización del algoritmo de búsqueda general en grafo:
  - Elegir para expandir el nodo menos profundo no expandido
  - Implementar la frontera como una cola FIFO, i.e., los nuevos nodos se insertan al final





# Búsqueda preferente en amplitud (“breadth-first”)

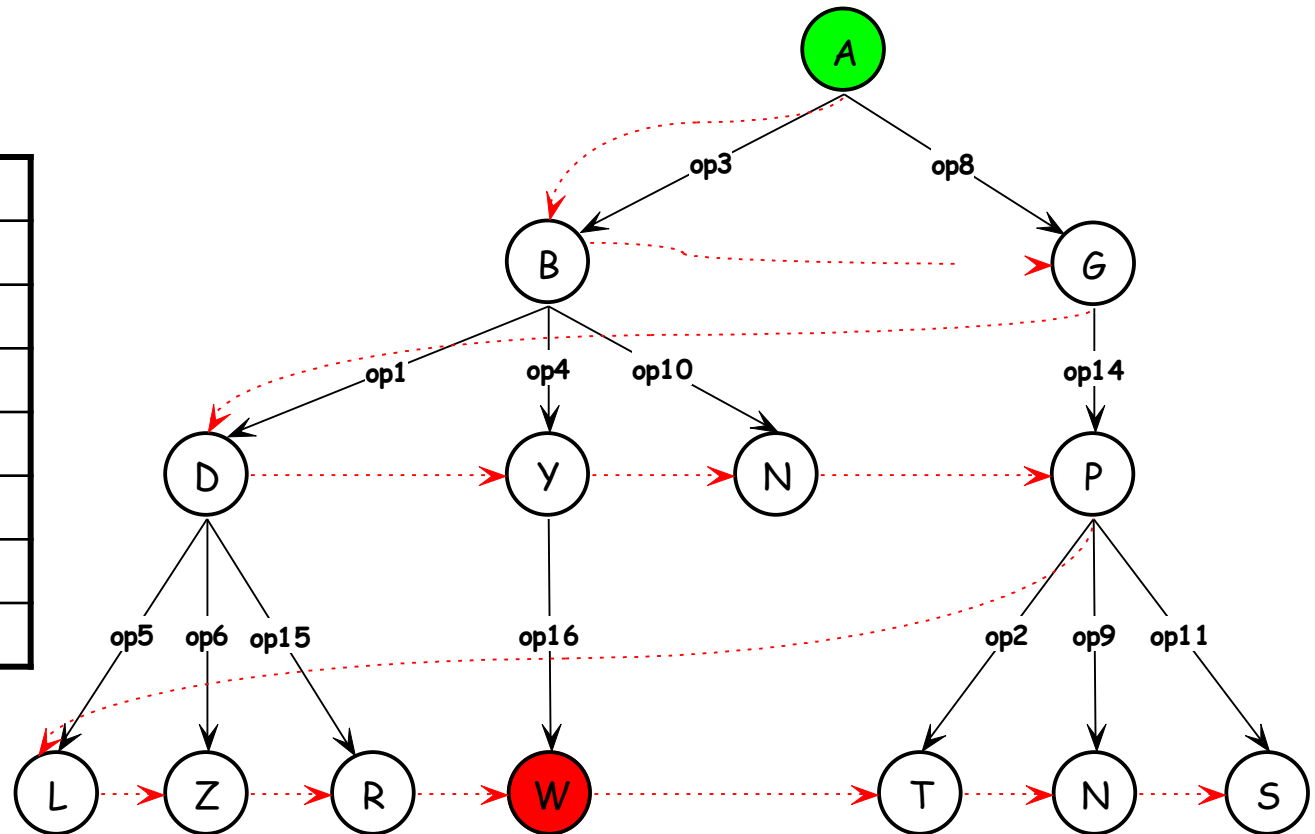


- cola ← camino conteniendo sólo la raíz
- mientras no está vacía la cola y la meta no se ha alcanzado hacer
  - Eliminar el primer nodo de la cola
  - Aplicar operadores relevantes
  - Crear los sucesores
  - Añadir los nuevos nodos al final de la cola
- Si se ha alcanzado la meta devolver éxito sino fallo

# Búsqueda preferente en amplitud (“breadth-first”)

- $I = [A]$ ;  $M = [W, F, K]$ ;

1	B -> D	9	P -> N
2	P -> T	10	B -> N
3	A -> B	11	P -> S
4	B -> Y	12	Q -> Y
5	D -> L	13	E -> F
6	D -> Z	14	G -> P
7	H -> V	15	D -> R
8	A -> G	16	Y -> W



# RESOLUCIÓN DE PROBLEMAS

- Procedimiento sistemático
- Solución aceptable
  - A 3 B 4 Y 16 W
- Otros caminos explorados
  - A 3 B 1 D 5 L
  - A 3 B 1 D 6 Z
  - A 3 B 1 D 15 R
  - A 3 B 10 N
  - A 8 G 14 P 2 T
  - A 8 G 14 P 9 N
  - A 8 G 14 P 11 S
- Caminos no resolutivos, pero no necesariamente desaprovechables
- Consume grandes recursos de memoria
- Es ineficiente temporalmente por la gran expansión de nodos
- Siempre encuentra la mejor solución, si existe, y si puede

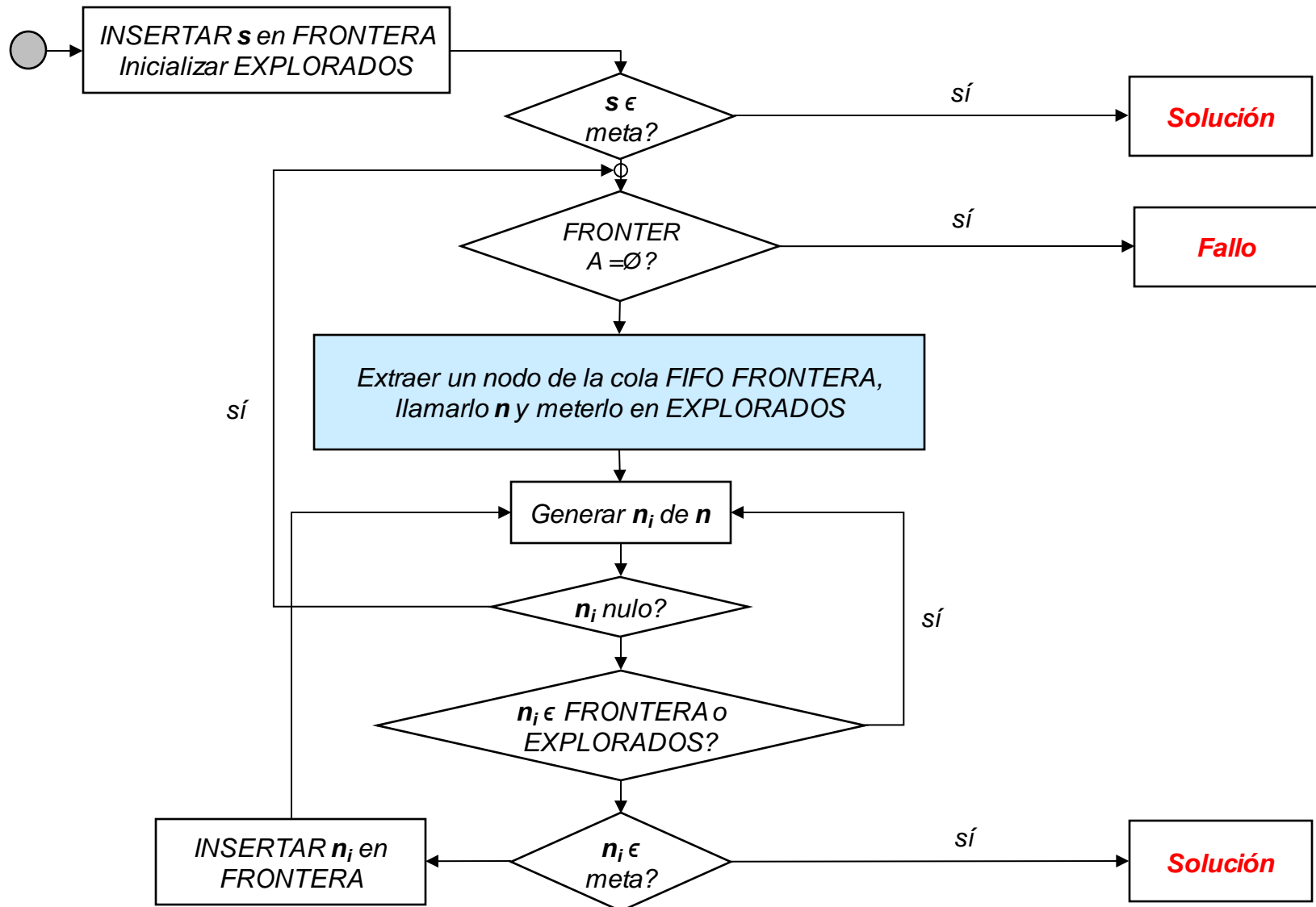
## Búsqueda preferente en amplitud

```
funcion BREADTH-FIRST-SEARCH (problema) devuelve solucion, o fallo
  nodo ← un nodo con ESTADO=problema.ESTADO-INICIAL
  COSTE-CAMINO=0

  si problema.TEST-META(nodo.ESTADO) entonces devuelve SOLUCION(nodo)
  FRONTERA ← cola FIFO con nodo como único elemento
  EXPLORADOS←conjunto vacío
  bucle hacer
    si VACIO?(FRONTERA) entonces devolver fallo
    sino nodo←POP(FRONTERA) /* nodo más superficial de la FRONTERA*/
    añadir nodo.ESTADO a EXPLORADOS
    para cada acción in problema.ACCIONES(nodo.ESTADO) hacer
      hijo←NODO-HIJO(problema, nodo, acción)
      si hijo.ESTADO no está en EXPLORADOS o FRONTERA entonces
        si problema.TEST-META(hijo.ESTADO)
          entonces devuelve SOLUCION(hijo)
        sino FRONTERA←INSERTAR(hijo,FRONTERA)
```

```
funcion NODO-HIJO (problema, padre, acción) devuelve un nodo
  devuelve un nodo con
    ESTADO = problema.RESULTADO(padre.ESTADO,accion)
    PADRE = padre, ACCION = accion
    COSTE-CAMINO = padre.COSTE-CAMINO +
      problema.COSTE-PASO(padre.ESTADO, accion)
```

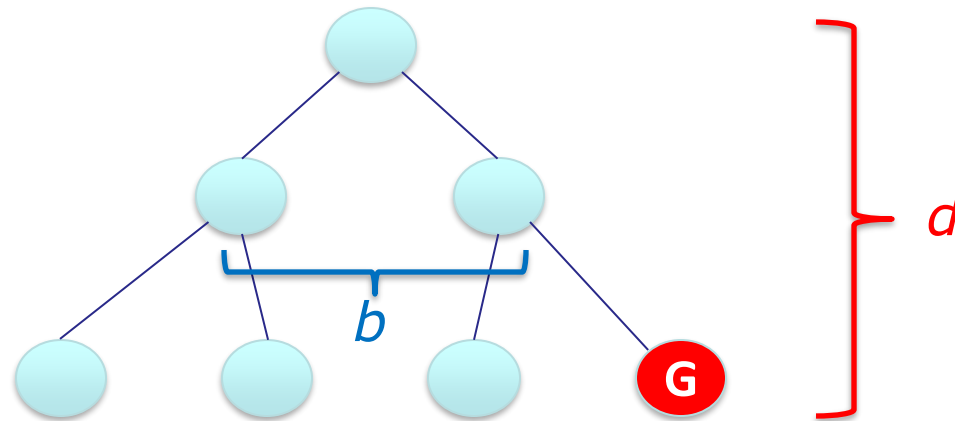
## Búsqueda preferente en amplitud



- Completa:
  - Sí, si el nodo meta está a una profundidad  $d$ , expande antes los nodos más próximos (suponiendo  $b$  finito)
- Óptima:
  - En general, no es óptima (el nodo meta más cercano no tiene por qué ser el óptimo)
  - Sí, todas las acciones tienen el mismo coste

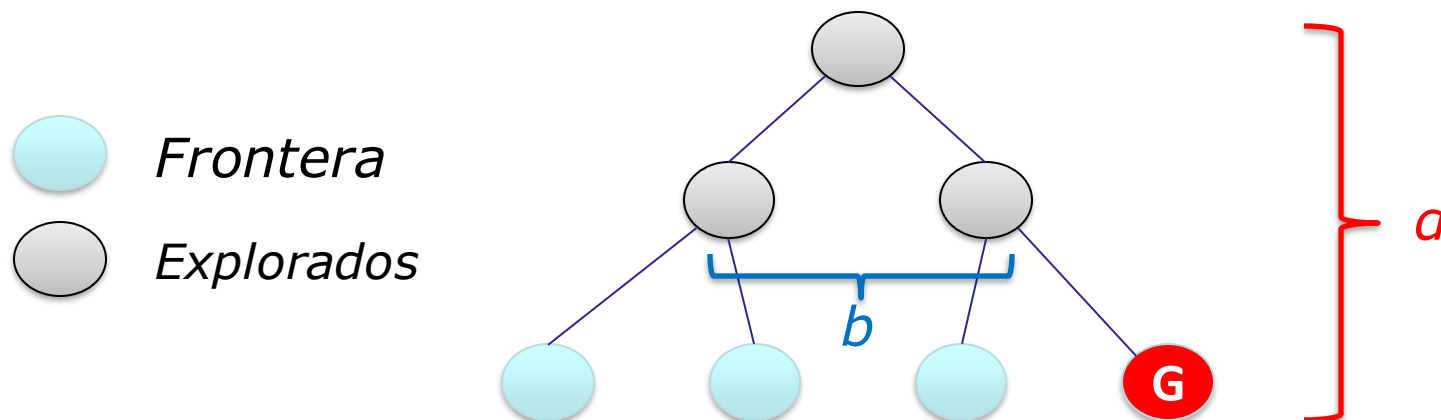
# Evaluación de la búsqueda preferente en amplitud

- Complejidad temporal:
  - Si la meta se encuentra a una profundidad  $d$ , hay que generar todos los nodos de ese nivel



- $b + b^2 + b^3 + \dots + b^d = O(b^d)$

- Complejidad espacial:
  - Cada nodo generado permanece en memoria. Al nivel del nodo meta se almacena la mayor cantidad de nodos:  $O(b^{d-1})$  en el conjunto de explorados y  $O(b^d)$  en la frontera.



- Así pues:  $b + b^2 + b^3 + \dots + b^{d-1} + b^d = O(b^d)$



# Estrategias de búsqueda no informada

## Evaluación de la búsqueda preferente en amplitud

Profundidad	Nodos	Tiempo	Espacio
2	110	0,11 mseg.	107 Kb
4	11.110	11 mseg.	10,6 MB
6	$10^6$	1,1 s.	1 GB
8	$10^8$	2 min.	103 GB
10	$10^{10}$	3 h.	10 TB
12	$10^{12}$	13 días	1 PB
14	$10^{14}$	3,5 años	99 PB
16	$10^{16}$	350 años	10 EB

*Petabyte*= $10^{15}$  bytes  
*Exabyte*= $10^{18}$  bytes

$b=10$ , 1 millón nodos/s. y 1.000 bytes/nodo

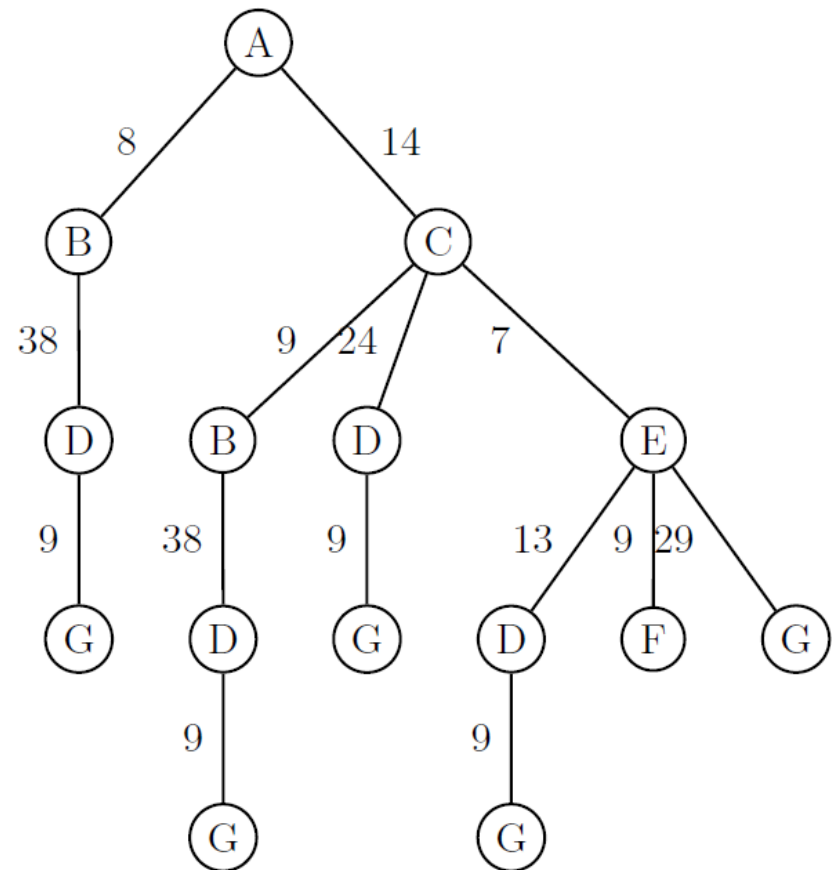
- Lecciones aprendidas:
  - Los requisitos de memoria son un problema mayor que el tiempo de ejecución
  - El tiempo todavía es un factor primordial

# Estrategias de búsqueda no informada

## Ejercicio

- Escribir la secuencia de generación y expansión de nodos.
- Suponer que la meta es G

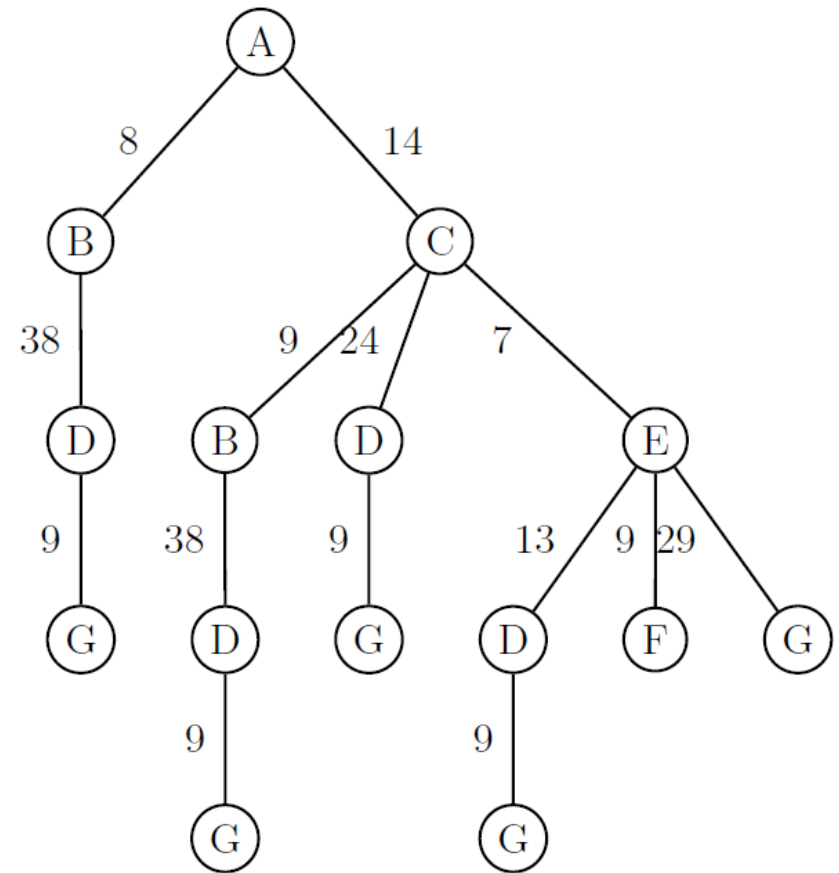
Paso	Frontera	Explorados
1	A	-
...	...	...



# Estrategias de búsqueda no informada

## Solución

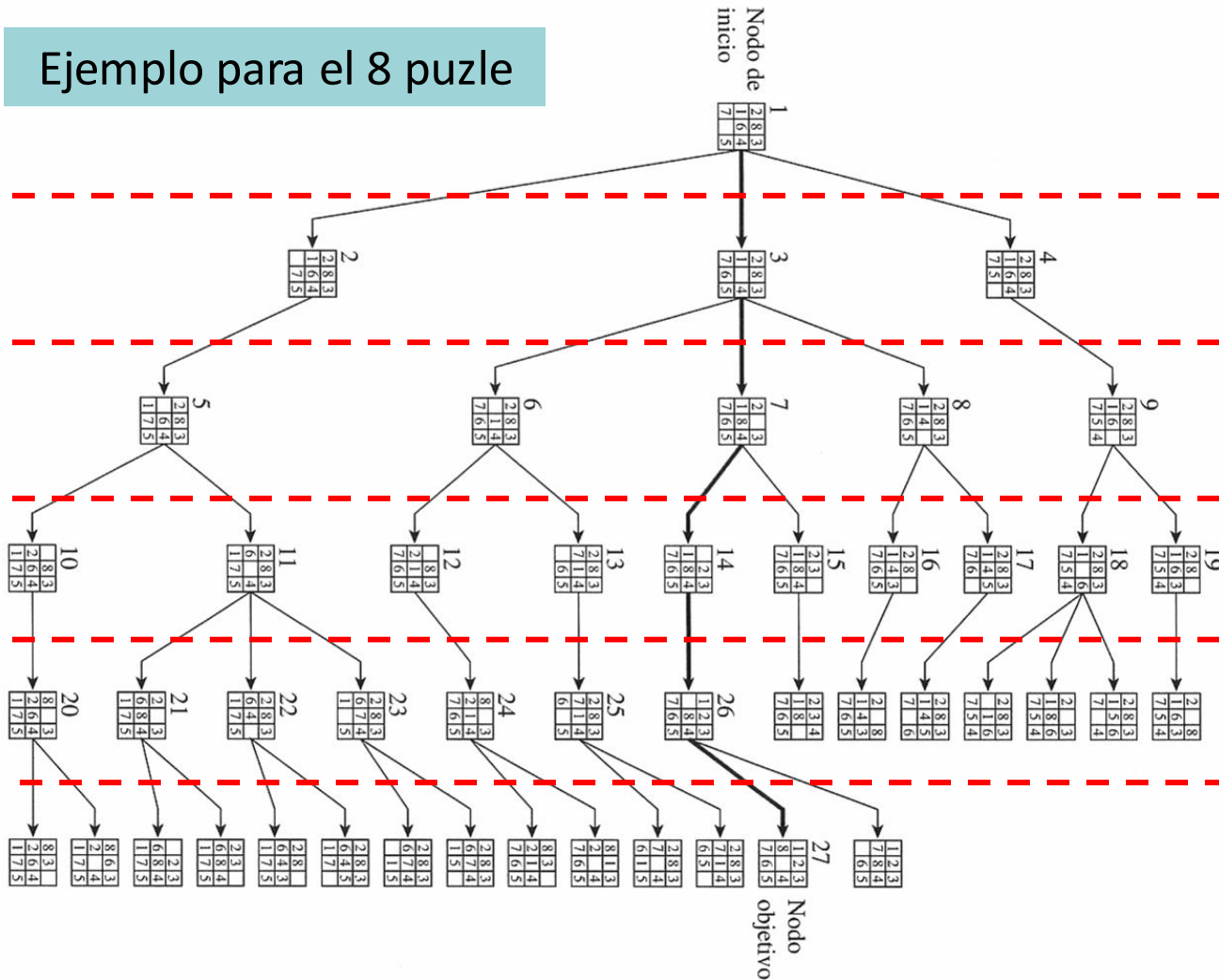
Paso	Frontera	Explorados
1	A	-
2	B,C	A
3	C,D	A,B
4	D,E	A,B,C
5	E,G	A,B,C,D



# Estrategias de búsqueda no informada

## Búsqueda preferente en amplitud

Ejemplo para el 8 puzle



# Búsqueda de coste uniforme

- La búsqueda primero en Anchura es óptima si los costes de transición son constantes
- Con una extensión sencilla podemos encontrar un algoritmo óptimo con cualquier función de coste por paso:
  - Expandir nodos con el menor coste de camino,  $g(n)$
  - Si todos los costes son iguales es idéntico a anchura
- Implementación:
  - *Frontera* es una cola de prioridad ordenada por  $g$
  - El *Test de meta* se aplica al nodo cuando se selecciona para su expansión, no tras su generación
  - Se comprueba si ya existe un camino mejor a un nodo en la frontera

# Estrategias de búsqueda no informada

## Búsqueda de coste uniforme

```
funcion UNIFORM-COST SEARCH(problema) devuelve una solucion o fallo
  nodo ← un nodo con ESTADO=problema.ESTADO-INICIAL,
  COSTE-CAMINO=0
  FRONTERA ← cola de prioridad ordenada por COSTE-CAMINO,
               con nodo como unico elemento
  EXPLORADOS ← Conjunto vacio

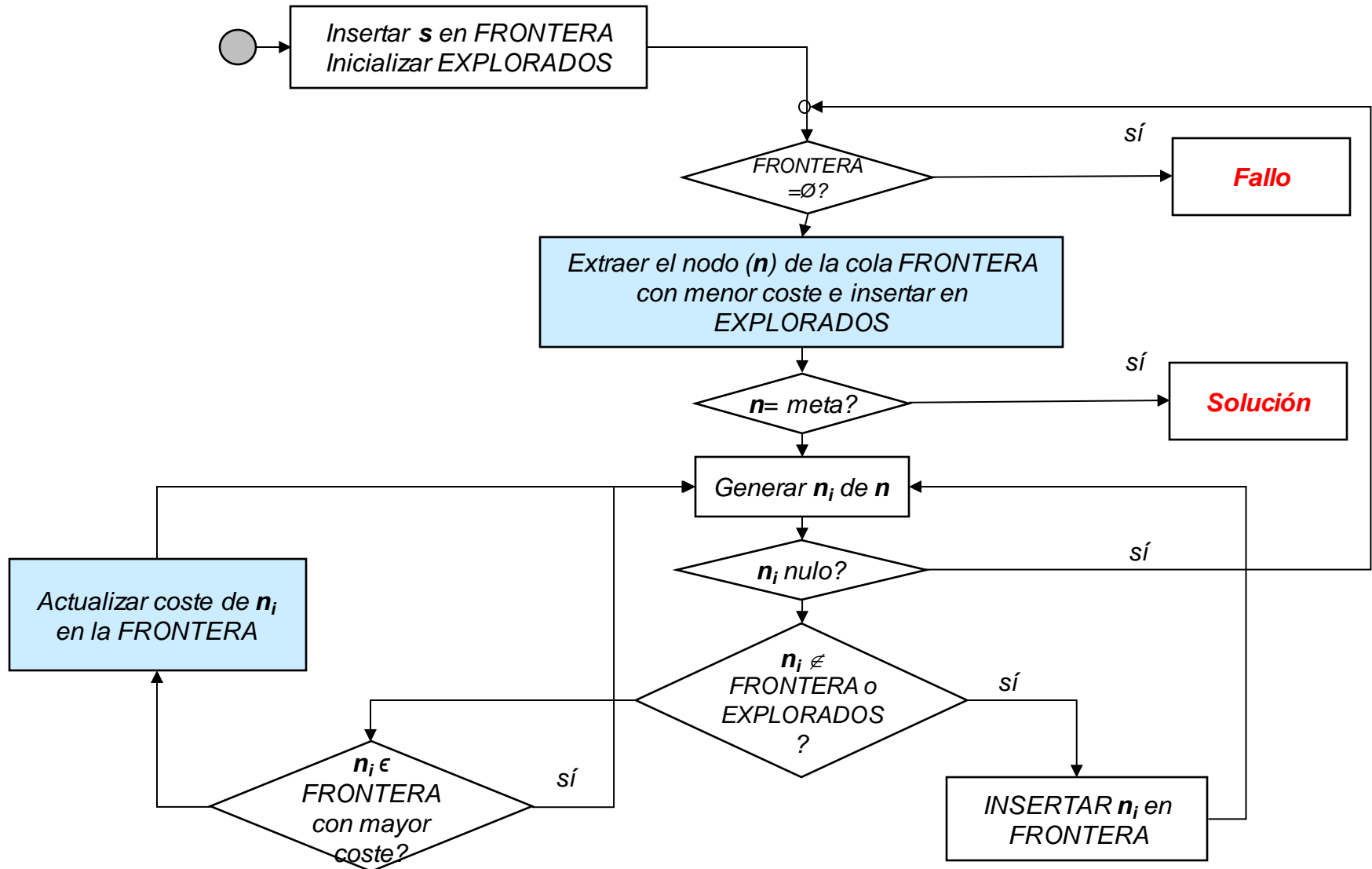
  bucle hacer
    si VACIA?(FRONTERA) entonces devolver fallo
    nodo ← POP(FRONTERA) /* elige el nodo con menor coste */
    si problema.TEST-META(nodo.ESTADO) entonces
      devuelve Solucion(nodo).

    añadir nodo.ESTADO a EXPLORADOS

  para cada accion en problema.ACCIONES(nodo.ESTADO) hacer
    hijo ← NODO-HIJO(problema, nodo, accion)
    si hijo.ESTADO no esta en EXPLORADOS o FRONTERA entonces
      FRONTERA ← Insertar(hijo, frontera)
    sino si hijo.ESTADO esta en FRONTERA con mayor COSTE-CAMINO
      entonces reemplazar ese nodo con hijo
```

# Estrategias de búsqueda no informada

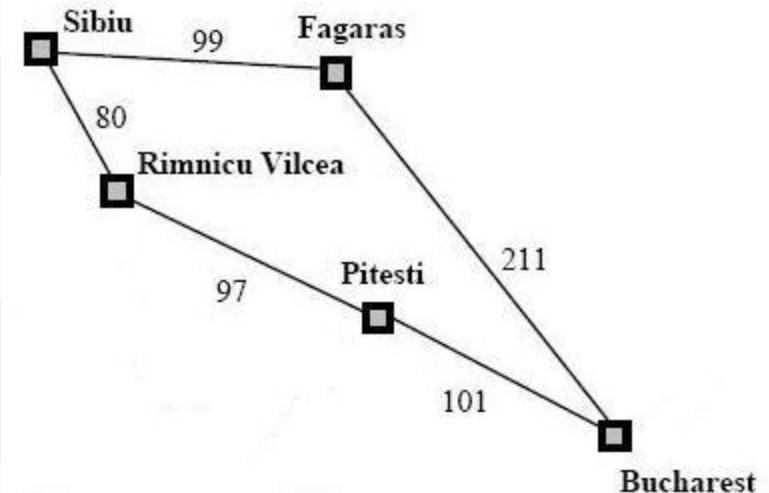
## Búsqueda de coste uniforme



# Estrategias de búsqueda no informada

## Búsqueda de coste uniforme

Paso	Frontera	Explorados
1	Sibiu (0)	-
2	Rimnicu Vilcea(80), Fagaras(99)	Sibiu(0)
3	Fagaras(99), Pitesti(177)	Sibiu(0), Rimnicu Vilcea(80)
4	Pitesti(177), Bucarest(310)	Sibiu(0), Rimnicu Vilcea(80), Fagaras(99)
5	Bucarest(278), <del>Bucarest(310)</del>	Sibiu(0), Rimnicu Vilcea(80), Fagaras(99), Pitesti(177)
	-	Sibiu(0), Rimnicu Vilcea(80), Fagaras(99), Pitesti(177), <b>Bucarest(278)</b>



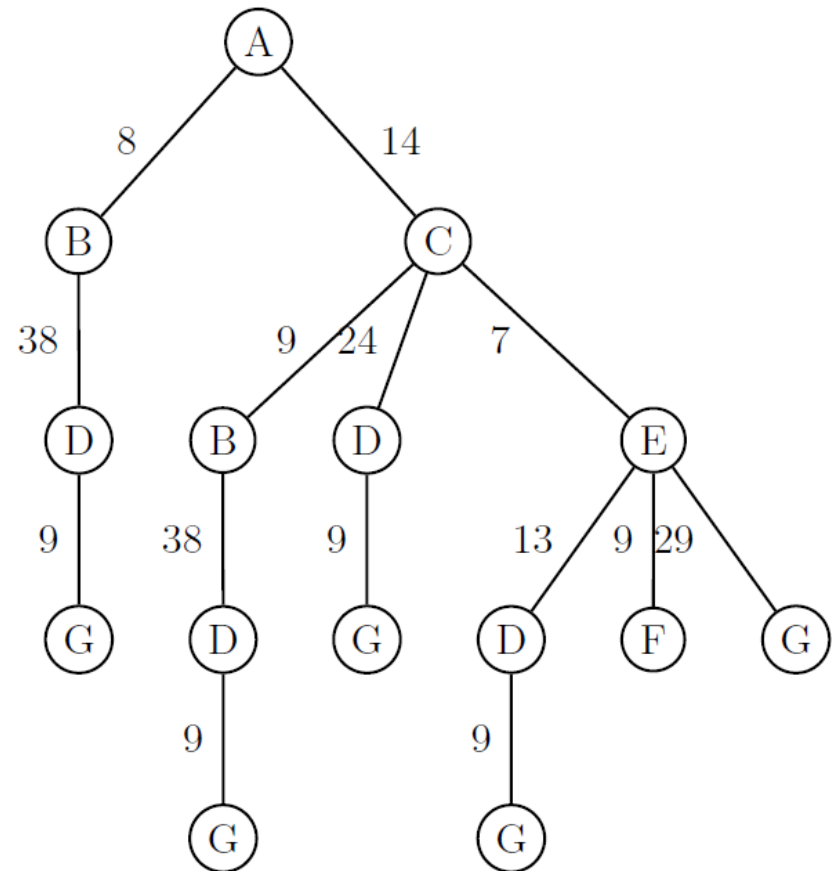


# Estrategias de búsqueda no informada

## Ejercicio

- Escribir la secuencia de generación y expansión de nodos.
- Suponer que la meta es G

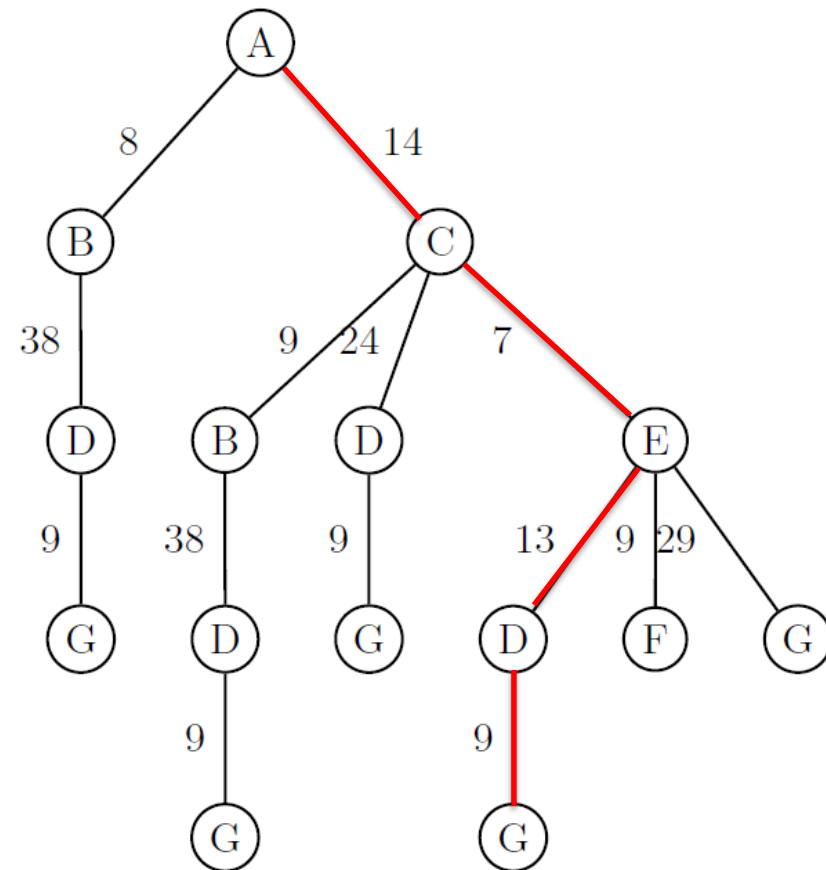
Paso	Frontera	Explorados
1	A	-



# Estrategias de búsqueda no informada

## Solución

Paso	Frontera	Explorados
1	A(0)	-
2	B(8), C(14)	A(0)
3	C(14), D(46)	A(0), B(8)
4	E(21), <del>B(23)</del> , D(38), <del>D(46)</del>	A(0), B(8), C(14)
5	D(34), <del>D(38)</del> , F(30), G(50)	A(0), B(8), C(14), E(21)
6	D(34), G(50)	A(0), B(8), C(14), E(21), F(30)
7	<del>G(50)</del> , G(43)	A(0), B(8), C(14), E(21), F(30), D(34)
		A(0), B(8), C(14), E(21), F(30), D(34), <b>G(43)</b>

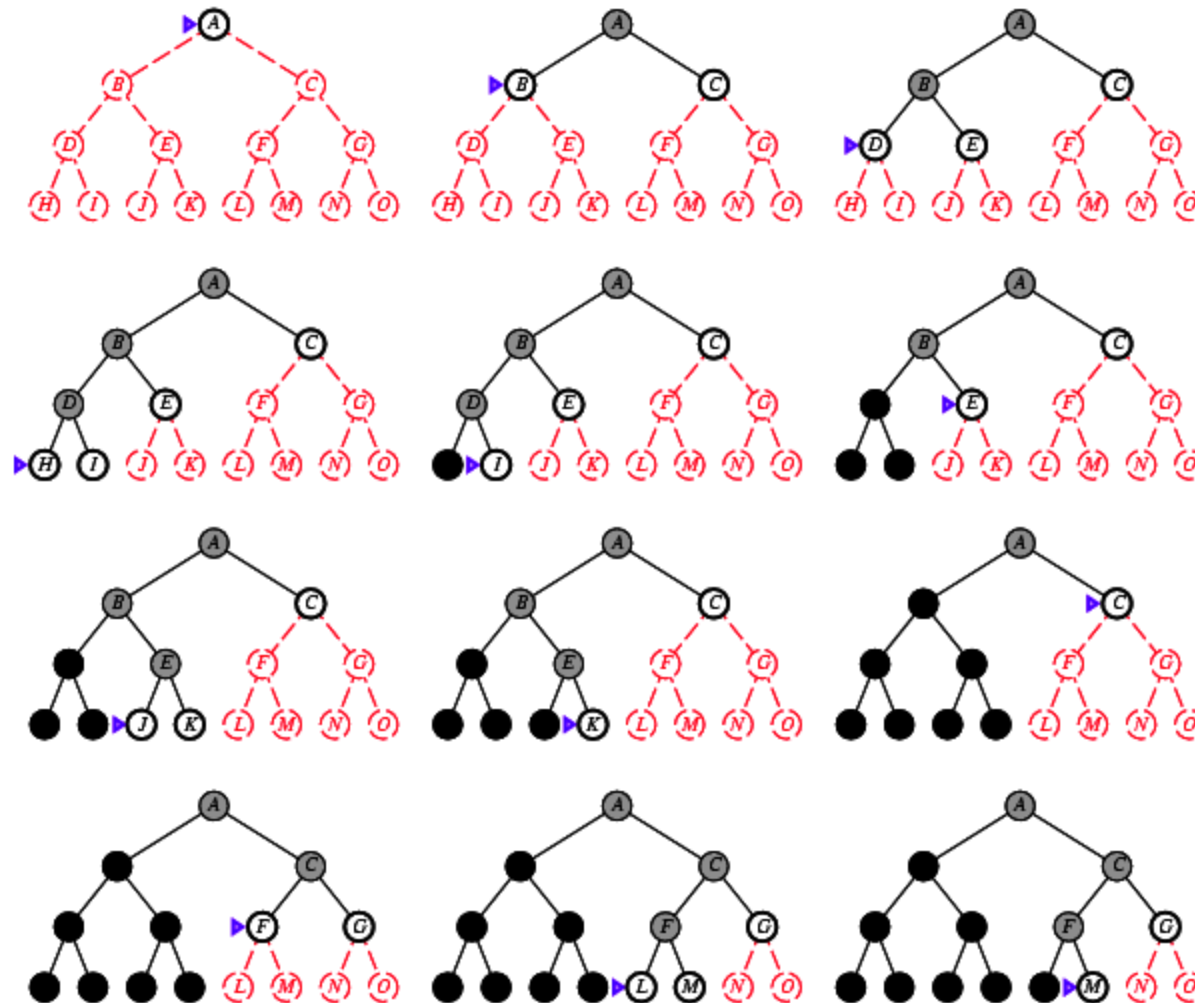


# Evaluación búsqueda de coste uniforme

- *Completa*: Sí, si el coste de cada paso es  $\geq E$  ( $E > 0$ )
- *Óptima*: nodos se expanden en orden creciente de coste (el primer nodo meta en ser expandido será el óptimo)
- *Complejidad*:
  - *Espacio-Temporal*: número de nodos con coste de camino  $\leq$  coste solución óptima  $O(b^{1+\lceil C^*/E \rceil})$
  - $C^*$  coste solución óptima, y  $E$  coste de cada acción
  - Si coste de los pasos es el mismo  $b^{1+\lceil C^*/E \rceil} = b^{d+1}$

- Expandir los nodos más profundos de la frontera actual del árbol
- Una vez generados, la búsqueda retrocede al siguiente nodo más profundo con sucesores sin explorar
- Particularización del algoritmo de búsqueda general en grafo:
  - Elegir para expandir el nodo más recientemente generado
  - Implementar la frontera como una cola LIFO, i.e., los nuevos nodos se insertan por el principio

## Búsqueda preferente en profundidad



## Búsqueda preferente en profundidad

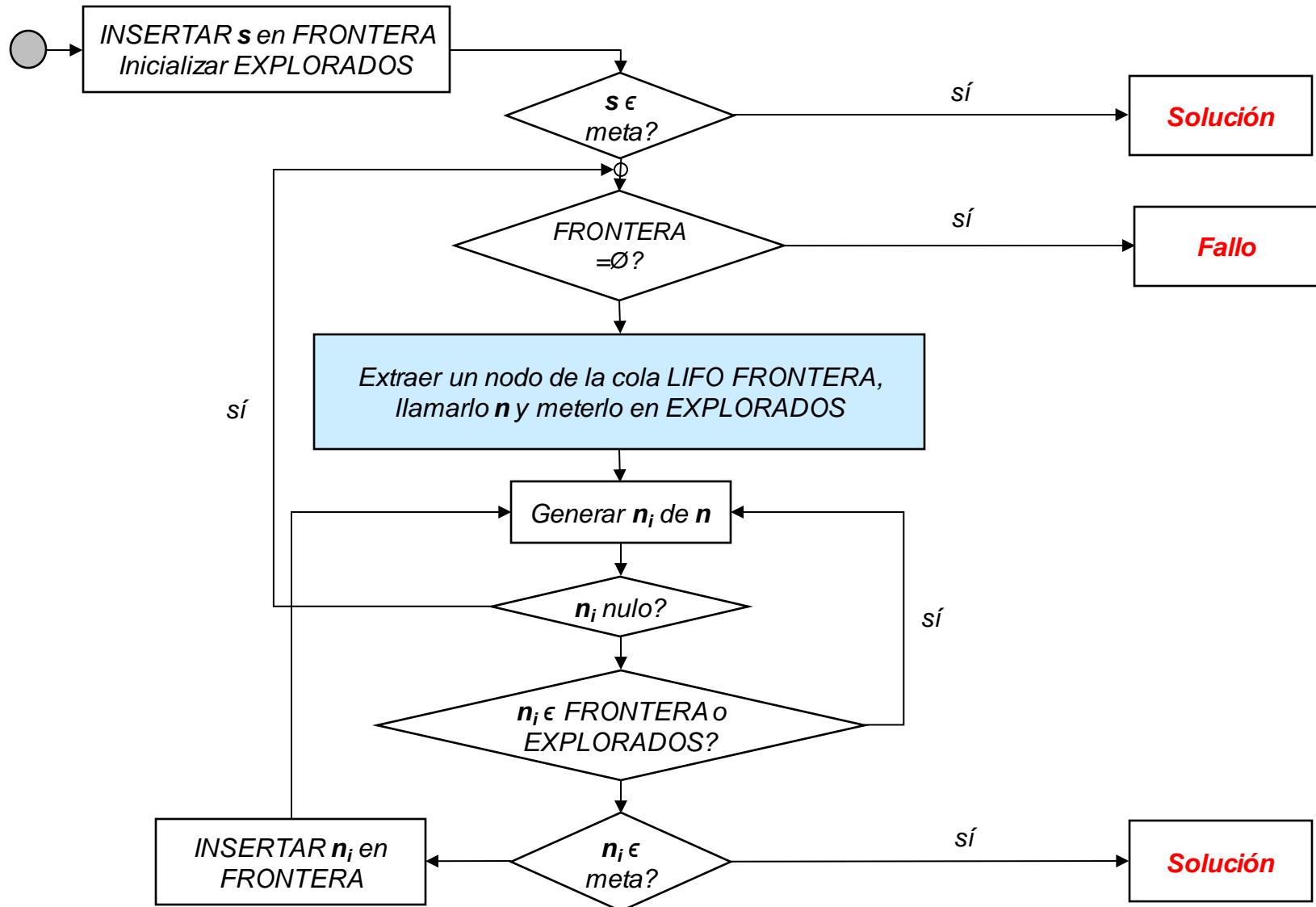
```
funcion DEPTH-FIRST-SEARCH (problema) devuelve solucion, o fallo
  nodo ← un nodo con ESTADO=problema.ESTADO-INICIAL
  COSTE-CAMINO=0

  si problema.TEST-META(nodo.ESTADO) entonces devuelve SOLUCION(nodo)
  frontera ← cola LIFO con nodo como único elemento

  bucle hacer
    si VACIO?(frontera) entonces devolver fallo
    sino nodo ← POP(frontera)
    para cada acción in problema.ACCIONES(nodo.ESTADO) hacer
      hijo ← NODO-HIJO(problema, nodo, acción)
      si problema.TEST-META(hijo.ESTADO)
      entonces devuelve SOLUCION(hijo)
      sino frontera ← INSERTAR(hijo, frontera)
```

```
funcion NODO-HIJO (problema, padre, acción) devuelve un nodo
  devuelve un nodo con
    ESTADO = problema.RESULTADO(padre.ESTADO, accion)
    PADRE = padre, ACCION = accion
    COSTE-CAMINO = padre.COSTE-CAMINO +
      problema.COSTE-PASO(padre.ESTADO, accion)
```

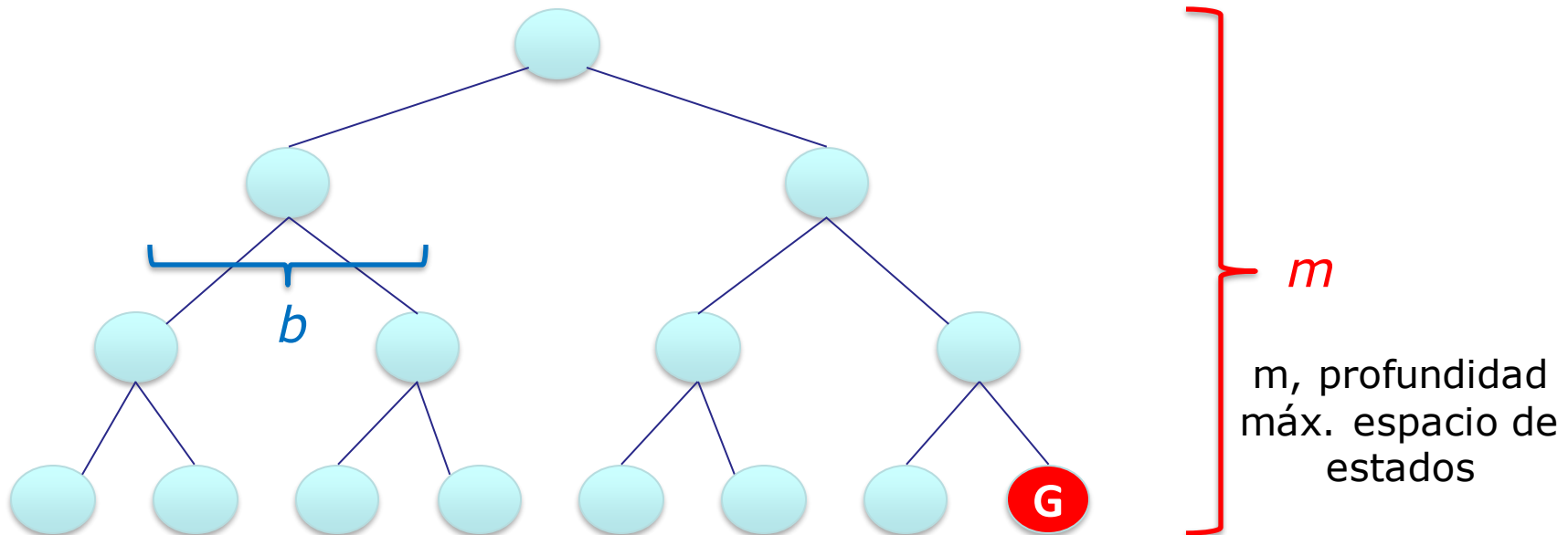
## Búsqueda preferente en profundidad



- Completa:
  - Si, si usamos búsqueda basada en grafos en espacios de estado finitos porque expandirá todos los nodos
  - No, si usamos búsqueda basada en árboles
  - No, en espacios de búsqueda infinitos si encuentran un camino infinito que no lleve a la meta
- Óptima:
  - No, puede encontrar una solución más profunda que otra en una rama no expandida



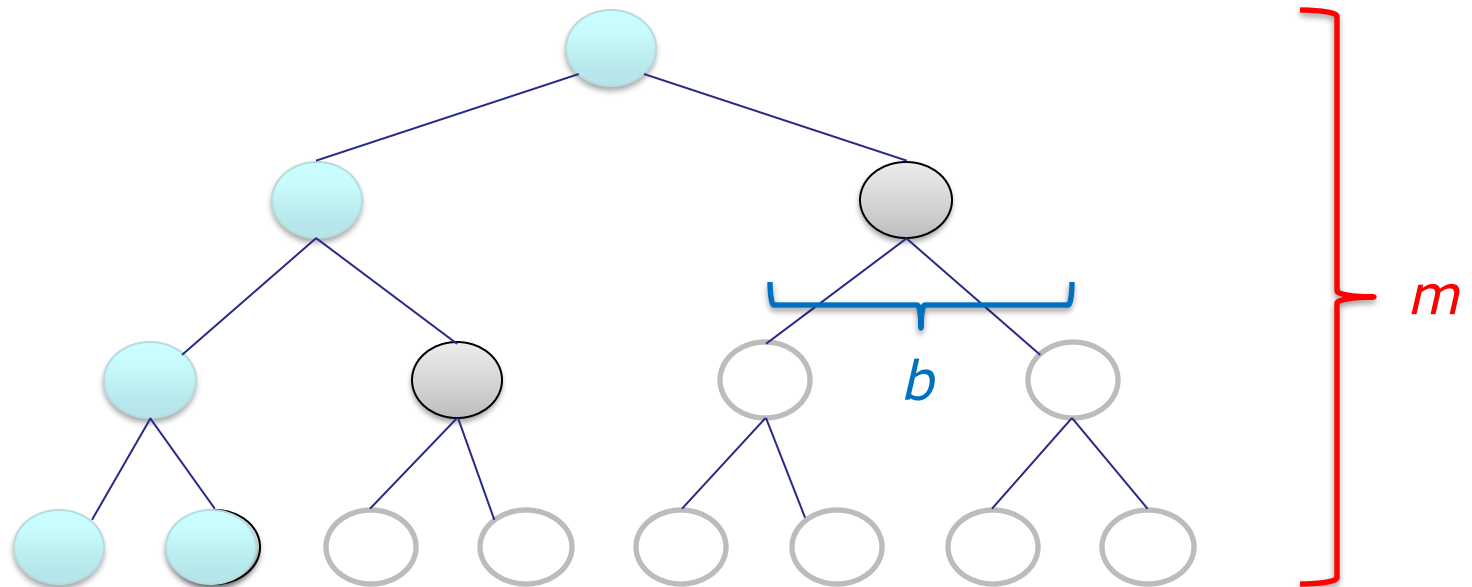
- Complejidad temporal:
  - Similar al tamaño del espacio de estados, tal vez infinito (búsqueda basada en grafo)
  - En el peor caso,  $b^m$  nodos :  $b^m + b^{m-1} + \dots + 1 = O(b^m)$  (búsqueda basada en árbol)



# Evaluación búsqueda preferente en profundidad

- Complejidad espacial:

- El mayor número de nodos almacenados se alcanza en el nodo inferior de más a la izquierda
- Sólo almacena un camino (raíz-hoja) para búsqueda en árbol
- En general:  $1+b+b+\dots^m\dots+b = 1+b*m = O(bm)$

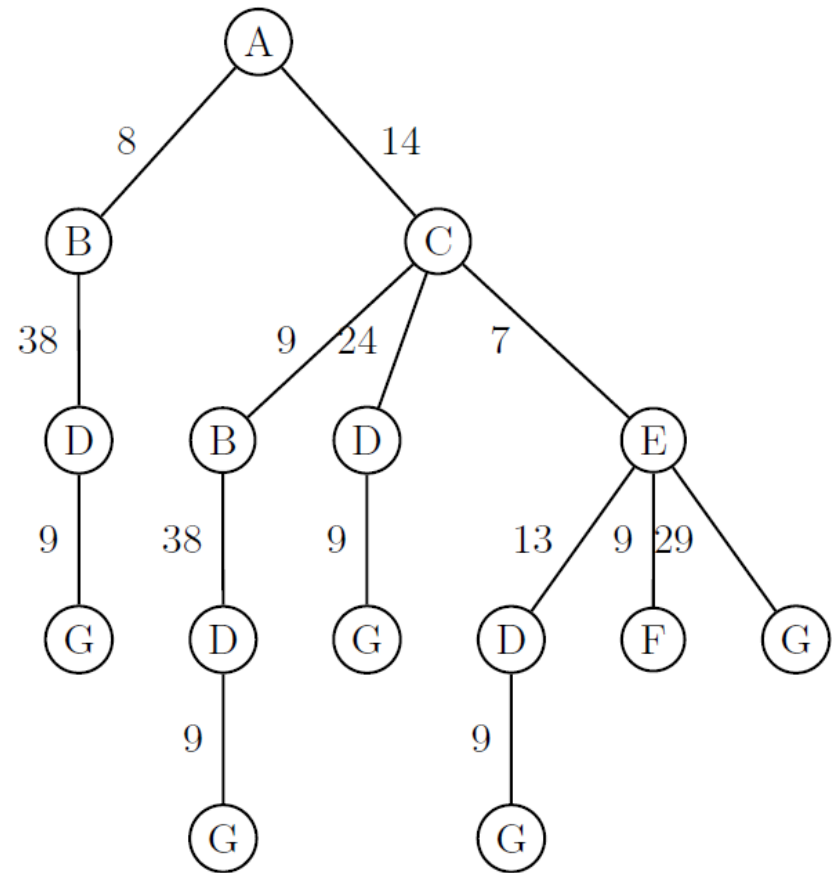


# Estrategias de búsqueda no informada

## Ejercicio

- Escribir la secuencia de generación y expansión de nodos.
- Suponer que la meta es G

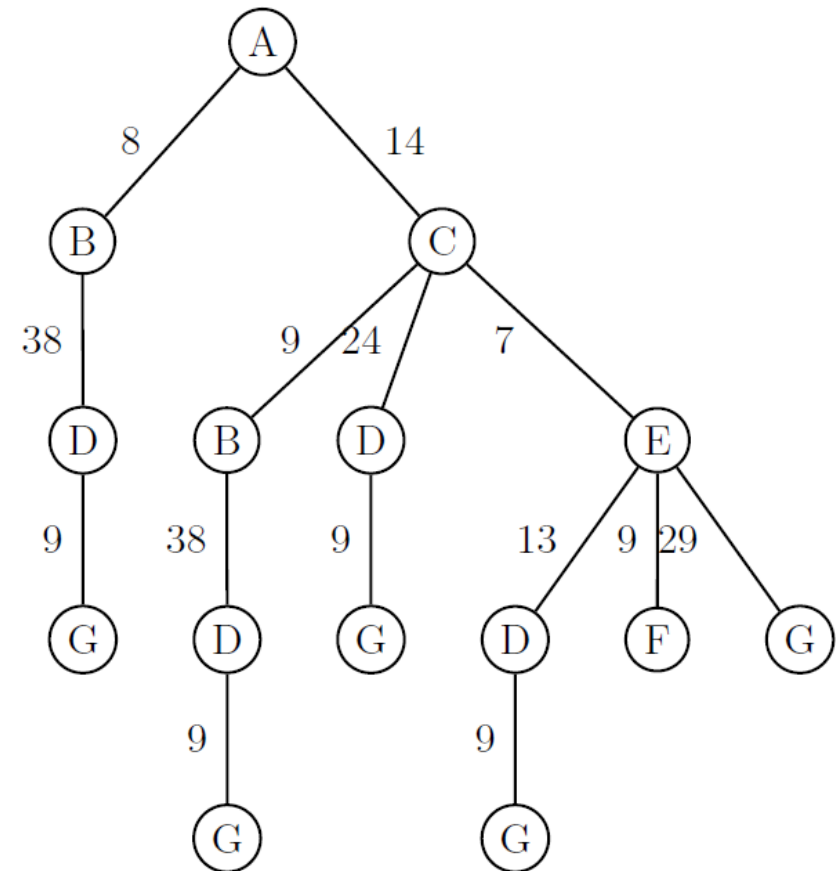
Paso	Frontera	Explorados
1	A	-
...	...	...



# Estrategias de búsqueda no informada

## Solución

Paso	Frontera	Explorados
1	A	-
2	B,C	A
3	D,C	A,B
4	G,C	A,B,D



## Búsqueda preferente en profundidad

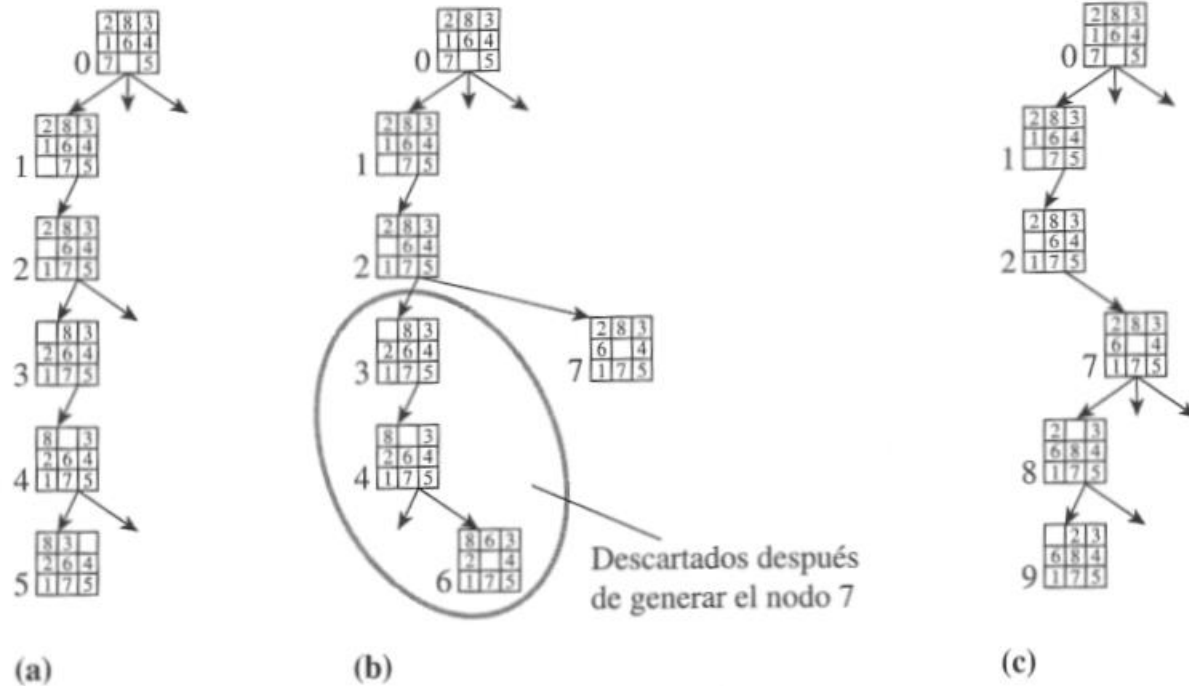
- Variante: Búsqueda con *backtracking*
  - Usa menos memoria todavía
  - Sólo se genera un sucesor de cada vez
  - Cada nodo parcialmente expandido recuerda cuál es el siguiente sucesor a generar
  - Complejidad espacial:  $O(m)$  en vez de  $O(b^*m)$
  - Otro truco añadido para ahorrar memoria y tiempo.
    - Generar el sucesor modificando la descripción del estado actual
    - Requisitos de memoria: 1 descripción de estado y  $O(m)$  acciones

# Búsqueda preferente en profundidad

- Usar...
  - La profundidad se conoce o está limitada
  - Las soluciones se encuentran a la misma profundidad
  - Cualquier solución sirve
  - La velocidad es prioritaria
- Evitar usar...
  - Subárboles infinitos o grandes
  - Se requieren las soluciones menos profundas

# Búsqueda preferente en profundidad

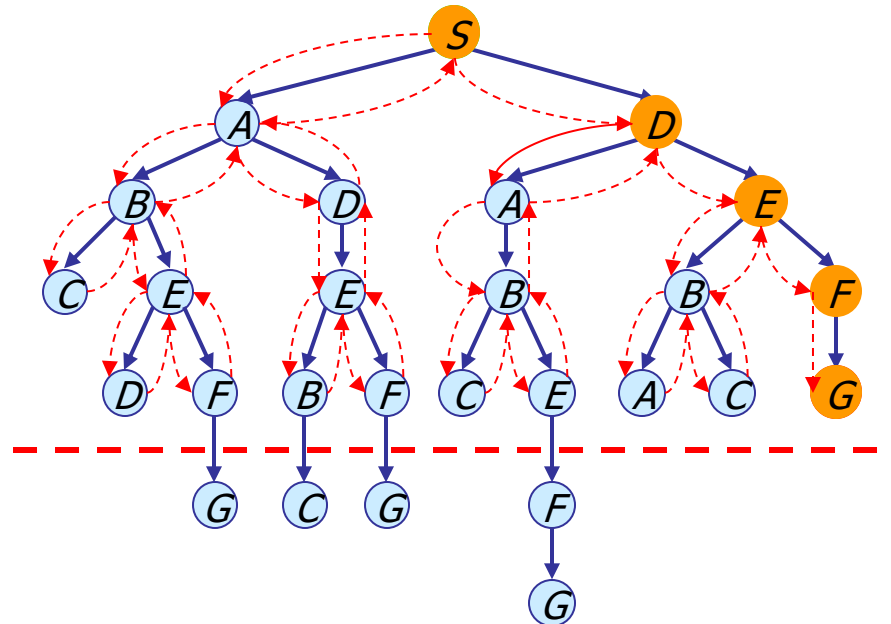
## Ejemplo para el 8 puzle



# Estrategias de búsqueda no informada

## Búsqueda de profundidad limitada

- fijar un límite a la profundidad máxima
- Soluciona caminos infinitos
- Si  $l < d$  (meta más allá del límite) incompleto
- Si  $l > d$  no es óptimo.
- Complejidad temporal  $O(b^l)$  y espacial  $O(b \cdot l)$





# Estrategias de búsqueda no informada

## Búsqueda de profundidad limitada

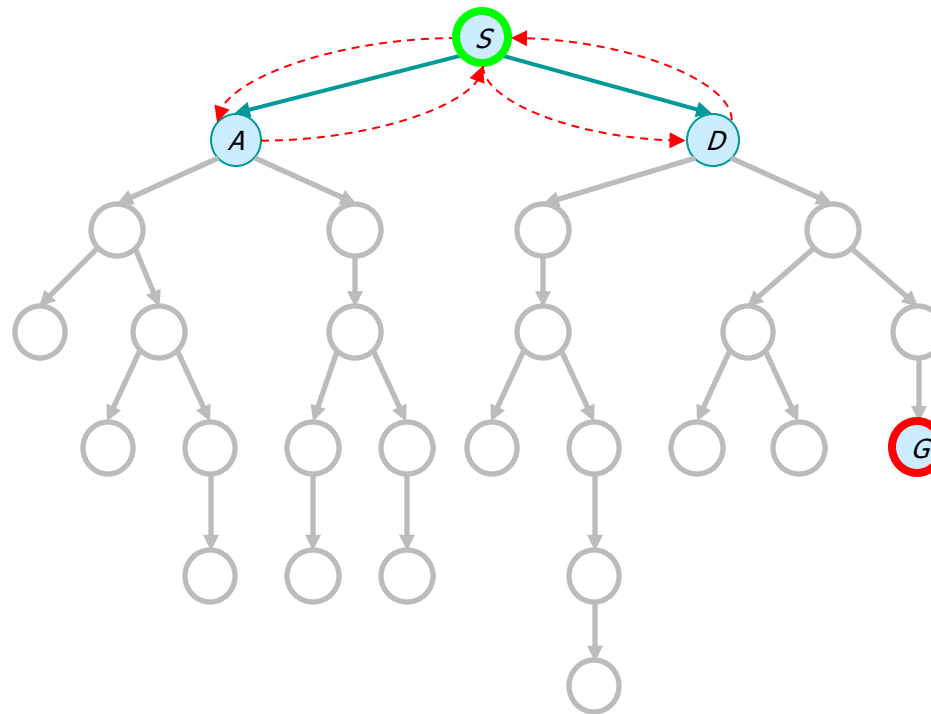
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
    cutoff_occurred?  $\leftarrow$  false
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      result  $\leftarrow$  RECURSIVE-DLS(child, problem, limit - 1)
      if result = cutoff then cutoff_occurred?  $\leftarrow$  true
      else if result  $\neq$  failure then return result
  if cutoff_occurred? then return cutoff else return failure
```

**Figure 3.17** A recursive implementation of depth-limited tree search.

## Búsqueda de profundidad iterativa

- Variante con profundidad iterativa
  - Incrementar gradualmente el límite (i.e. 0, 1, 2, ...)
  - Beneficios de la búsqueda en anchura y en profundidad
  - En espacios de búsqueda muy grandes en los que no se conoce la profundidad de la solución

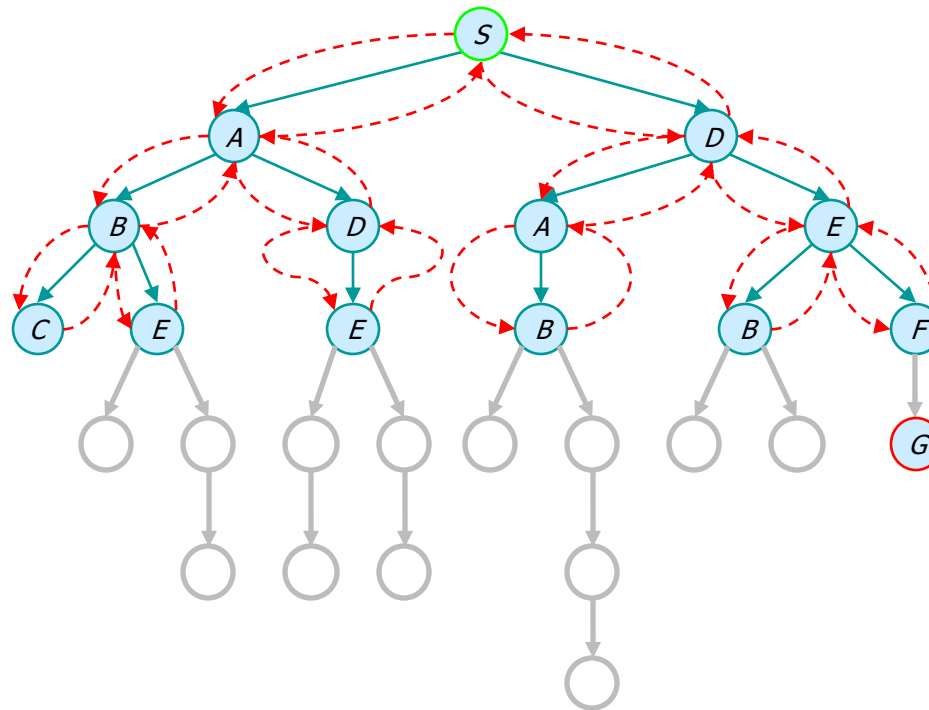


- 
- The diagram shows a hierarchical tree structure. The root node is 'S' (green). It has two children: 'A' (left) and 'D' (right). Node 'A' has children 'B' and 'D'. Node 'D' has children 'A' and 'E'. Node 'E' has children 'A' and 'G' (red). The tree is composed of nodes and edges, with some nodes highlighted in green or red.

# Estrategias de búsqueda no informada

## Búsqueda de profundidad iterativa

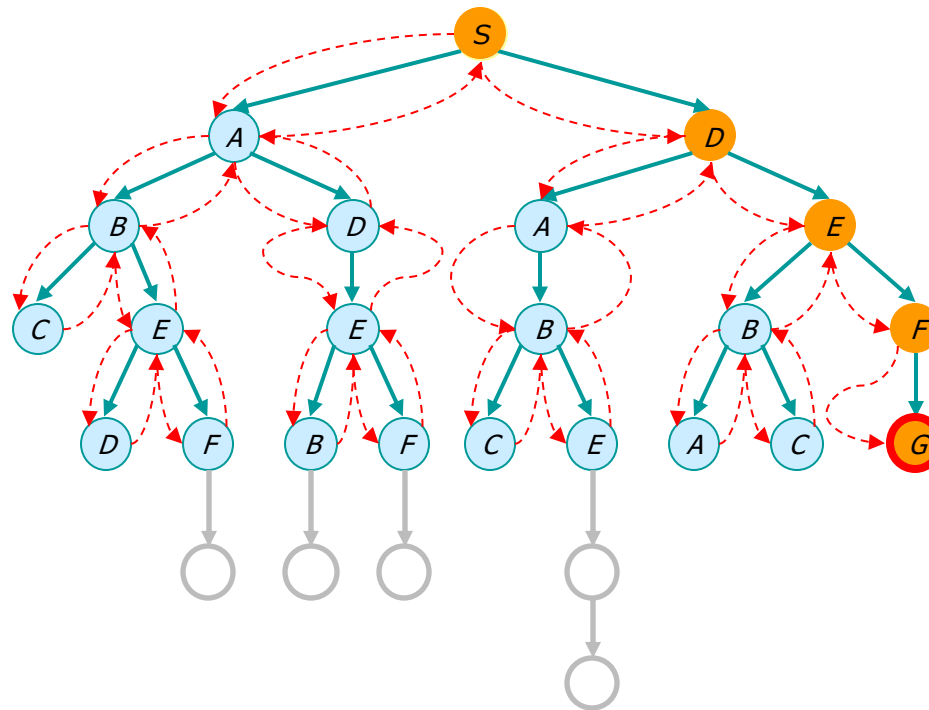
- Variante con profundidad iterativa
  - Incrementar gradualmente el límite (i.e. 0, 1, 2, ...)
  - Beneficios de la búsqueda en anchura y en profundidad
  - En espacios de búsqueda muy grandes en los que no se conoce la profundidad de la solución



# Estrategias de búsqueda no informada

## Búsqueda de profundidad iterativa

- Variante con profundidad iterativa
  - Incrementar gradualmente el límite (i.e. 0, 1, 2, ...)
  - Beneficios de la búsqueda en anchura y en profundidad
  - En espacios de búsqueda muy grandes en los que no se conoce la profundidad de la solución



# Estrategias de búsqueda no informada

## Búsqueda de profundidad iterativa

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

**Figure 3.18** The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

# Estrategias de búsqueda no informada

## Comparativa para búsqueda en árboles

	Amplitud	Coste Uniforme	Profundidad	Profundidad limitada	Profundización iterativa
<b>Completa</b>	Si <sup>a</sup>	Si <sup>a,b</sup>	No	Si ( $l \geq d$ )	Si <sup>a</sup>
<b>Óptima</b>	Si (coste cte)	Si	No	No	Si <sup>c</sup>
<b>Complejidad Espacial</b>	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(bd)$
<b>Complejidad Temporal</b>	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$

b: factor de ramificación

d: profundidad de la solución menos profunda

m: máxima profundidad del árbol de búsqueda

l: límite de profundidad

<sup>a</sup> completa si b es finito

<sup>b</sup> completa si coste de paso  $\geq \epsilon$ ,  $\epsilon > 0$

<sup>c</sup> óptima si los costes de transición son iguales



## Estrategias de búsqueda informada (búsqueda heurística)





### 1. **Búsqueda preferente por el mejor:**

- Basados en la búsqueda en anchura
- Examinar primero nodos que, de acuerdo con la heurística, están situados en el mejor camino hacia el objetivo.

### 2. **Búsqueda local:**

- Tratar de mejorar el estado actual en lugar de explorar de manera sistemática los caminos desde el estado inicial.
- El coste del camino es irrelevante y lo único importante es alcanzar el estado meta.

En ambos casos se posee conocimiento acerca de si un estado no meta es más prometedor que otro estado dado

# Búsqueda preferente por el mejor

- Basada en el algoritmo de búsqueda en árbol o grafo
- El nodo a expandir se selecciona en base a la **función de evaluación,  $f(n)$** : primero el menor (mejor) valor
- La mayoría de estos algoritmos incluyen como componente de  $f$  una **función heurística  $h(n)$** :
  - Coste estimado del camino menos costoso desde el estado en el nodo  $n$  hasta el estado meta

- Funciones heurísticas:
  - Funciones de carácter numérico que nos permiten estimar el beneficio de una determinada transición en el espacio de estados
  - Definidas sobre las descripciones de los estados
  - Restricción: si  $n$  es un nodo *meta*, entonces  $h(n)=0$
  - Propósito:
    - Permiten optimizar los procesos de búsqueda
    - Guían la exploración del espacio de estados de la forma más provechosa posible
    - Sugieren el mejor camino a priori, cuando disponemos de varias alternativas

## Búsqueda preferente por el mejor nodo

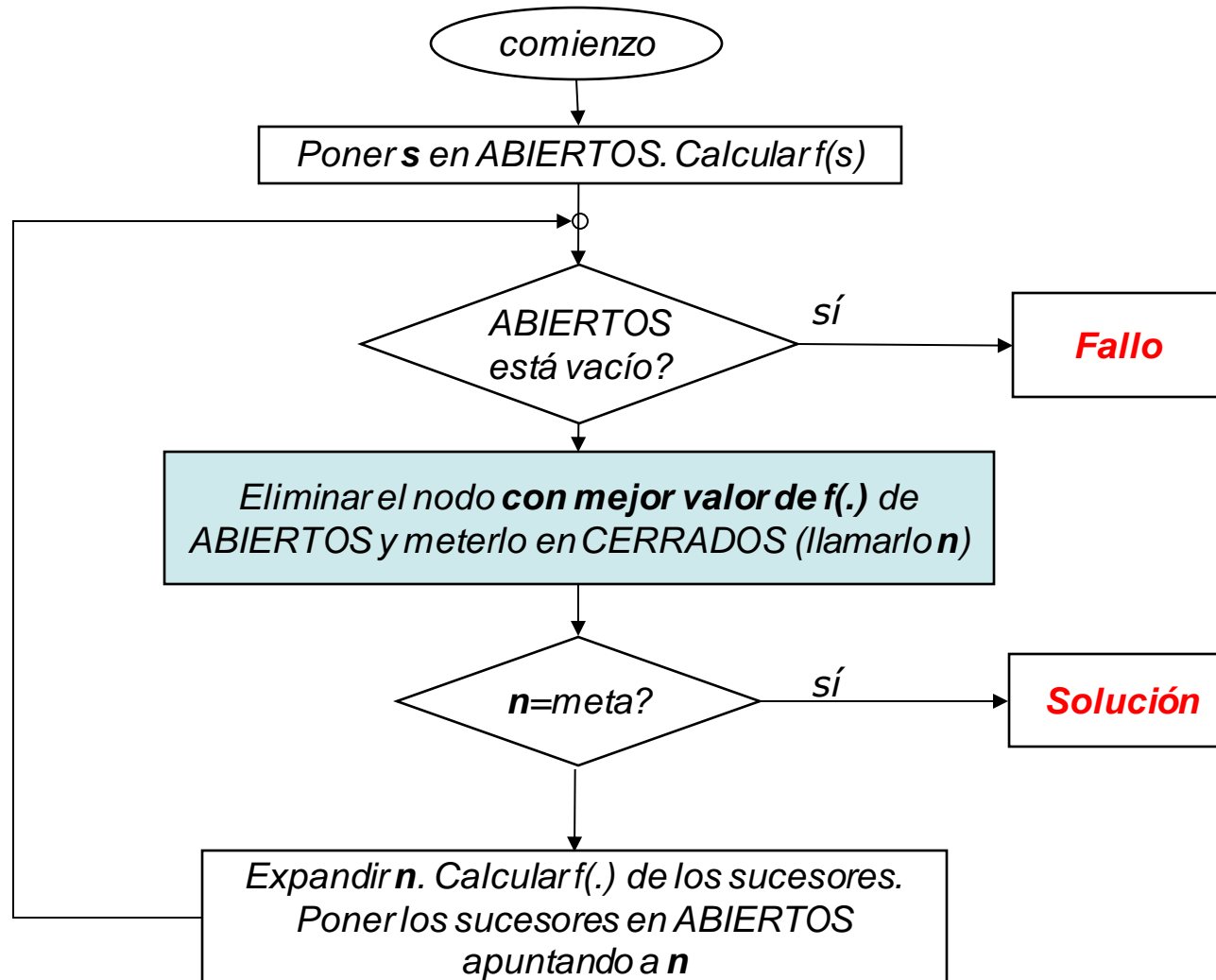
- Se selecciona un nodo para la expansión en base a una función de evaluación heurística,

*$f(n)$  mide la distancia al objetivo*

- Decide cuál es el mejor nodo a expandir
- $f(n)$  tomará valores pequeños en los nodos más prometedores
- Se expande el nodo  $n$  con la evaluación más baja de  $f$
- Se termina el proceso cuando el nodo a expandir sea un nodo objetivo
- Se implementa con una cola de prioridad que mantiene en orden ascendente la lista de nodos abiertos según el valor de  $f$

# Algoritmos de búsqueda informada

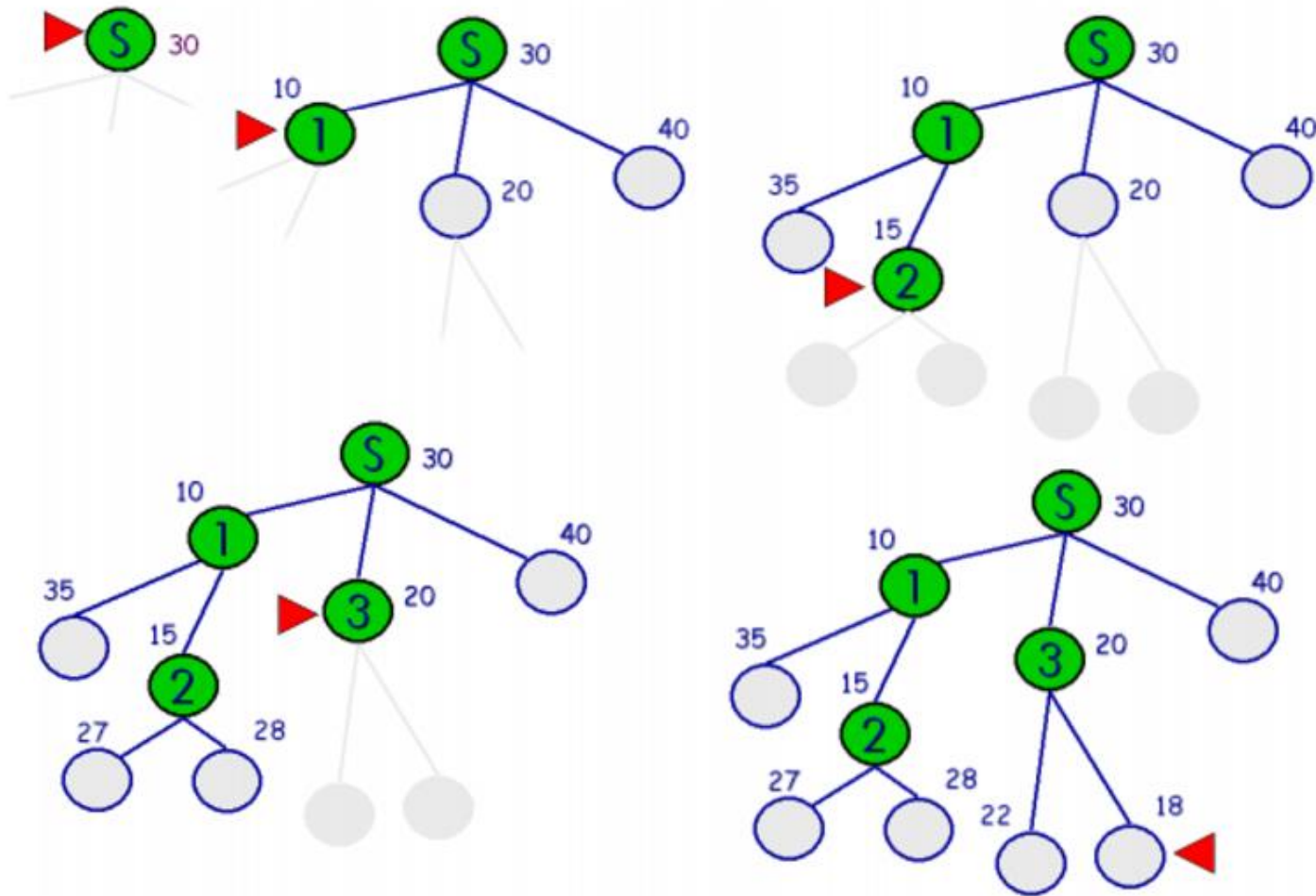
## Búsqueda preferente por el mejor nodo

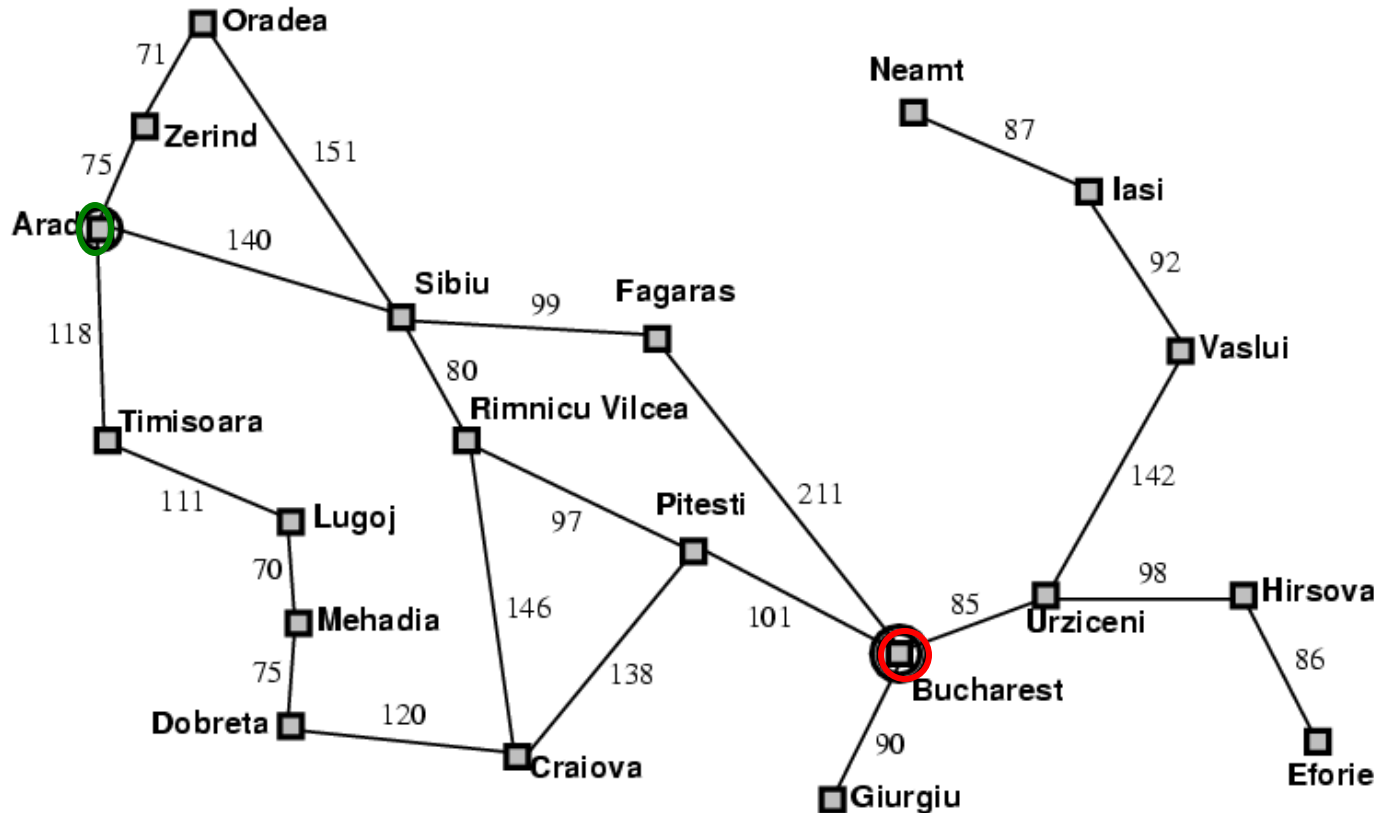


- Si  $f(n)$  sólo considera el coste mínimo estimado para llegar a una solución a partir de un nodo  $n$ ,  $h(n)$  o *función heurística*
  - $f(n) = h(n) \rightarrow$  **Búsqueda avara\***
  
- Si  $f(n)$  considera el coste mínimo total del camino a un nodo solución que pase por el nodo  $n$ 
  - $f(n) = g(n) + h(n) \rightarrow$  **Búsqueda A\* (A asterisco o estrella)**
  - $g(n)$  = coste consumido para llegar a  $n$
  
- $h(n) = 0$  si  $n$  es un nodo meta

# Algoritmos de búsqueda informada

## Búsqueda avara. Ejemplo





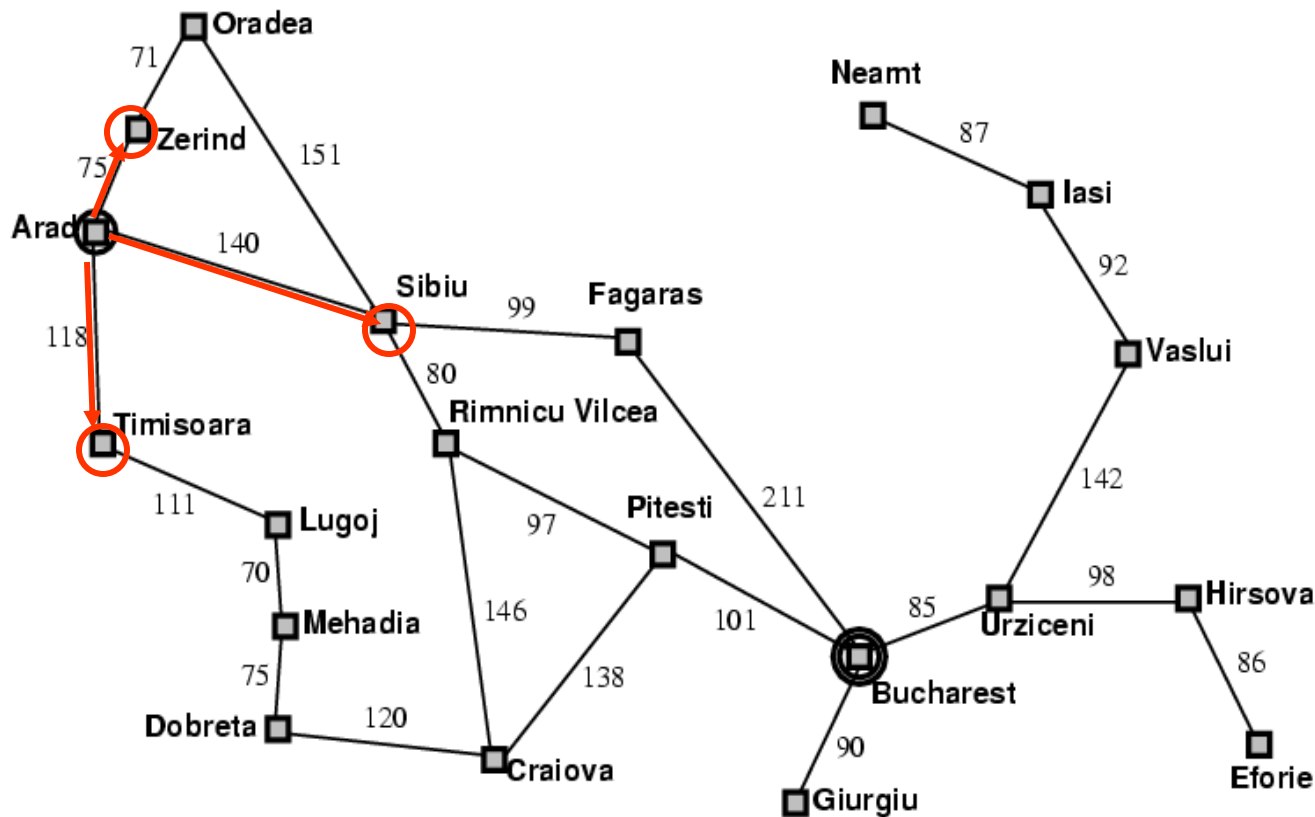
### Distancia en línea recta a Bucarest

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

Heurística= Distancia en línea recta



# Búsqueda avara. El problema de rutas en Rumanía



*Distancia en línea recta a Bucarest*

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

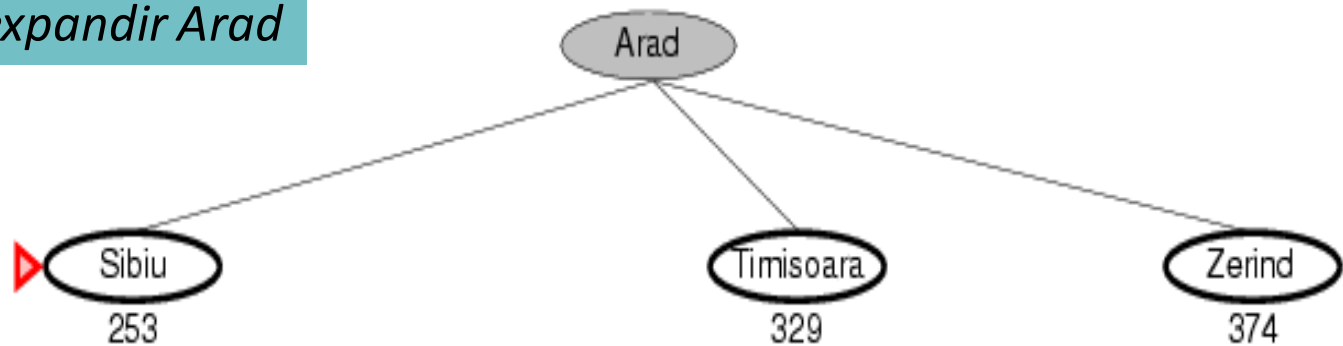
*Posibilidades:*

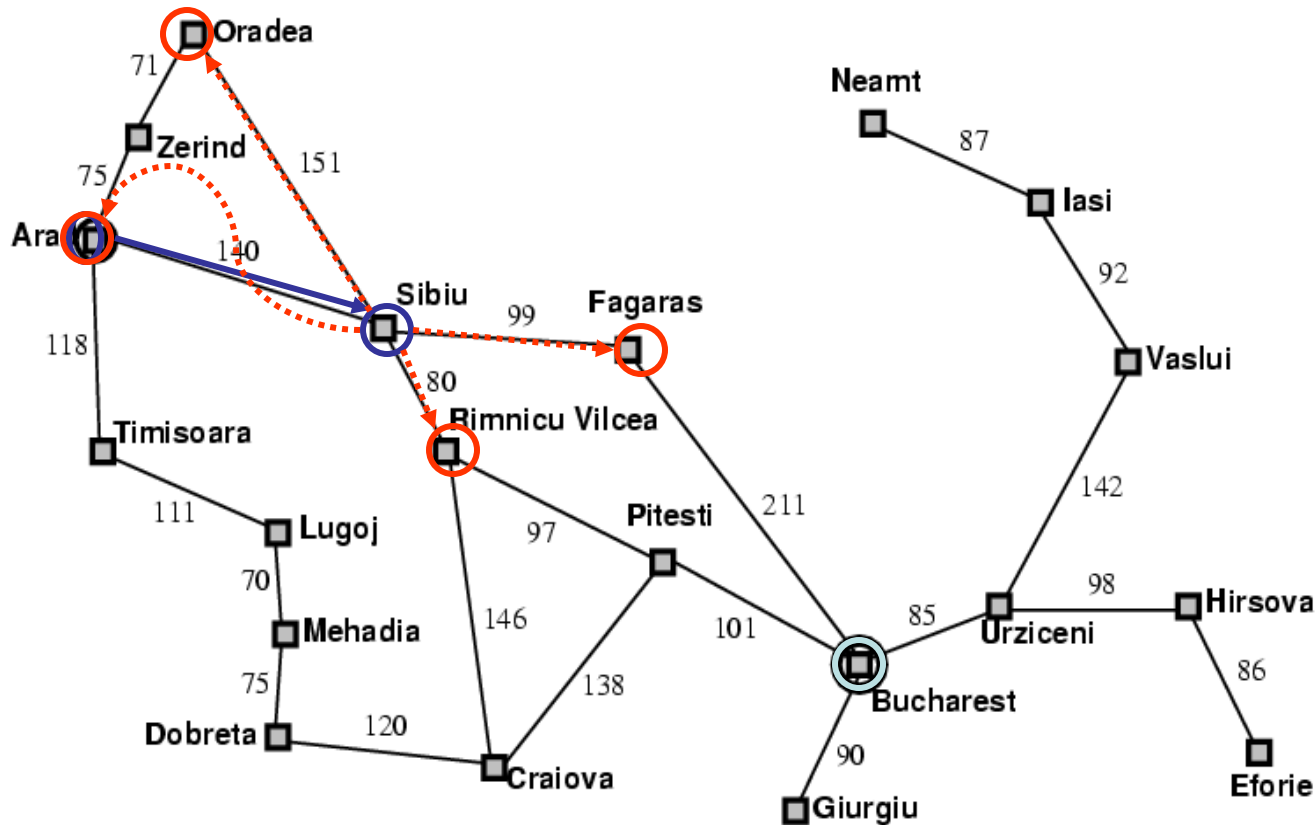
► *Sibiu (253), Timisoara (329), Zerind (374)*

*Estado inicial*



*Después de expandir Arad*





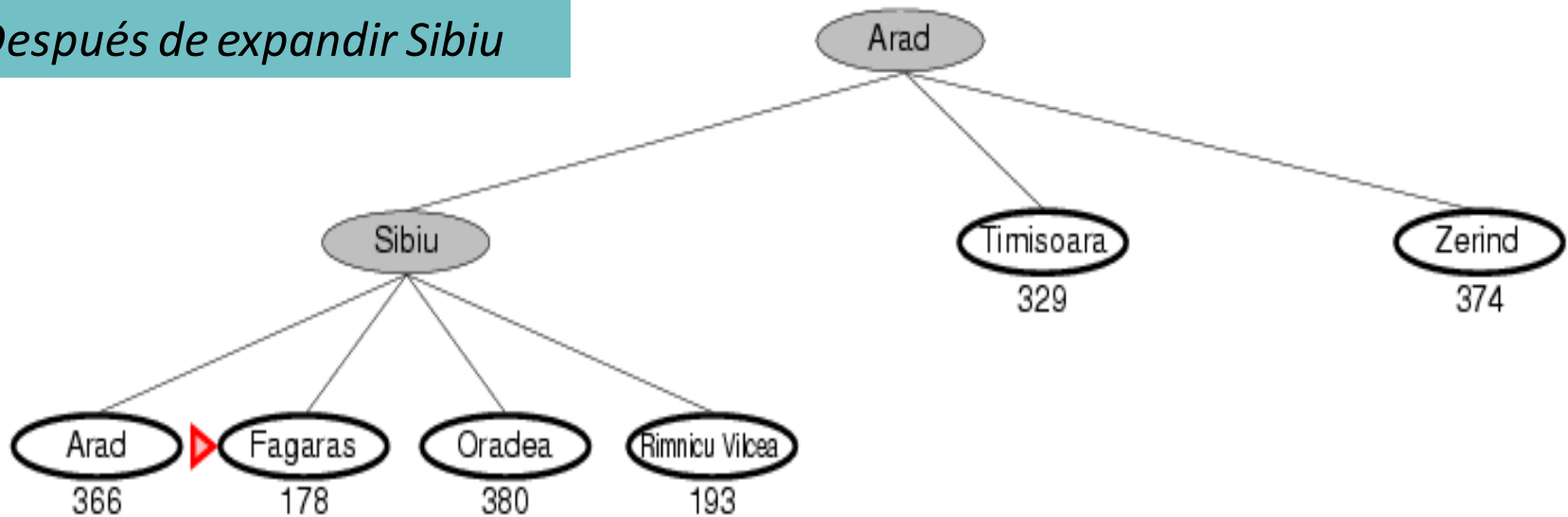
*Distancia en línea recta a Bucarest*

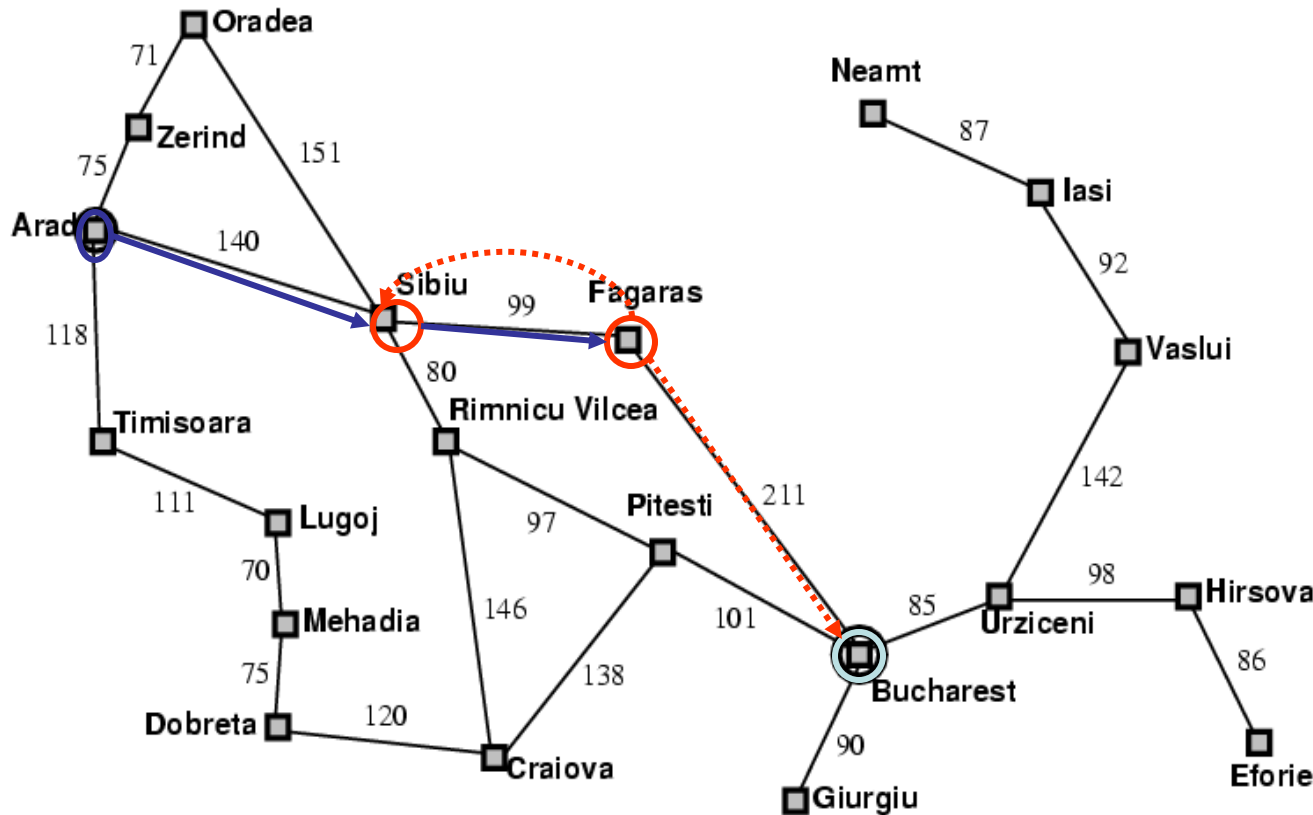
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

*Posibilidades:*

► *Fagaras (178), Oradea (380), Rimnicu Vilcea (193)*

*Después de expandir Sibiu*





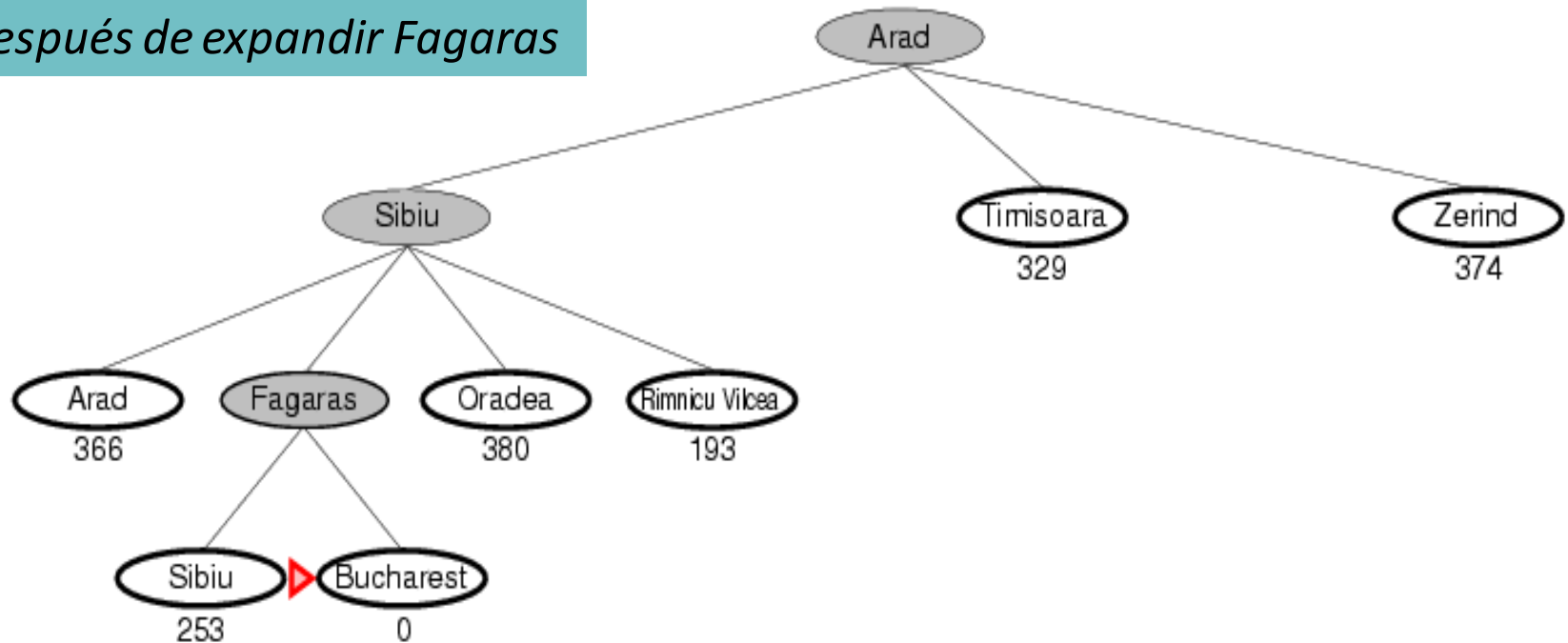
*Distancia en línea recta a Bucarest*

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

*Posibilidades:*

► *Bucharest (0)*

*Después de expandir Fagaras*

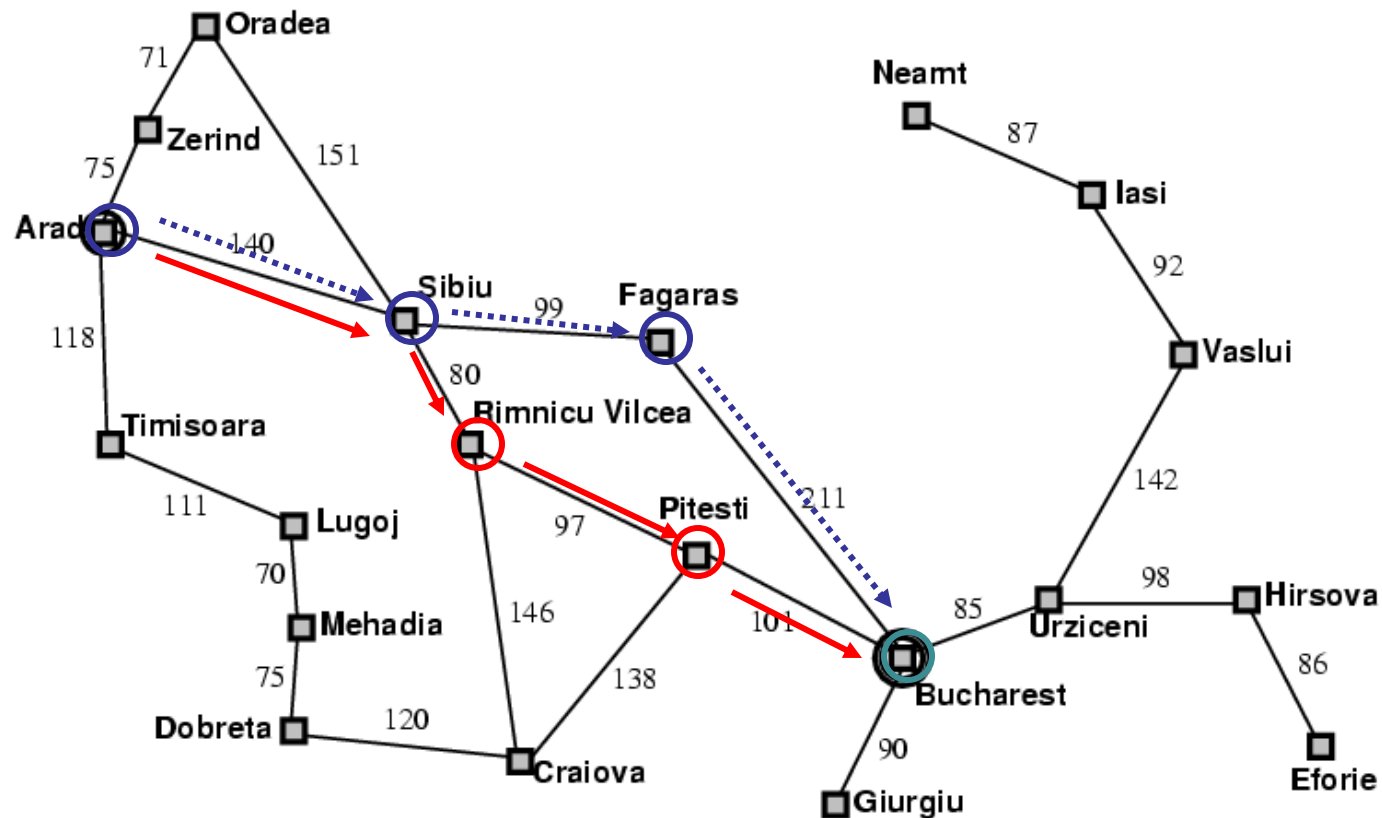


**Solución encontrada** Arad – Sibiu – Fagaras – Bucarest = 450

**Función de evaluación** Arad  $\rightarrow$  Bucarest = 366

# Algoritmos de búsqueda informada

## Búsqueda avara. El problema de rutas en Rumanía



.....► *Solución encontrada:* Arad – Sibiu – Fagaras – Bucarest = 450  
→ *Ruta óptima:* Arad - Sibiu - Rimniu - Pitesti – Bucharest = 418

- Evaluación de la estrategia de búsqueda

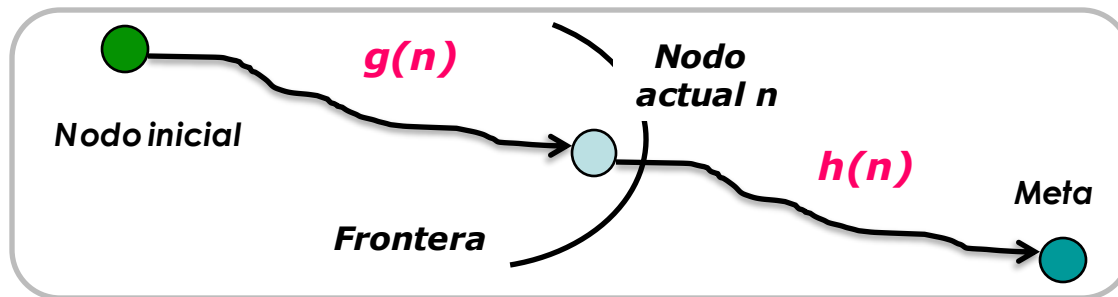
Búsqueda avara	
Óptima	No
Completa	No, puede perderse en bucles y recorrer rutas infinitas
Complejidad	<i>Temporal</i> $O(b^m)$ siendo m la profundidad máxima del árbol de búsqueda
	<i>Espacial</i> $O(b^m)$ mantienen todos los nodos en memoria



- Búsqueda avara:
  - minimizar el coste de alcanzar la meta,  $h(n)$
  - no es óptima ni completa
  - eficiente
- Búsqueda de coste uniforme:
  - minimizar el coste del camino,  $g(n)$
  - es óptima y completa
  - a veces muy ineficiente
- ¿Por qué no combinamos ambos métodos, en uno óptimo, completo y más eficiente?

## Búsqueda preferente por el mejor: A\*

- A\* o A-estrella es la forma más conocida de búsqueda preferente por el mejor
- Evalúa los nodos combinando  $g(n)$  y  $h(n)$ :
  - $g(n)$ , coste real del mejor camino para alcanzar el nodo  $n$
  - $h(n)$ , coste estimado del camino menos costoso desde el nodo  $n$  a la meta
  - $f(n) = g(n) + h(n) \rightarrow$  coste estimado de la solución menos costosa que pasa a través del nodo  $n$



# Estrategias de búsqueda no informada

## Búsqueda A\*

```
funcion A* SEARCH(problema) devuelve una solucion o fallo
  nodo ← un nodo con ESTADO=problema.ESTADO-INICIAL,
  F-EVALUACION= problema.HEURISTICA(nodo.ESTADO)
  COSTE-CAMINO=0
  FRONTERA ← cola de prioridad ordenada por menor F-EVALUACION,
               con nodo como unico elemento
  EXPLORADOS ← Conjunto vacio

  bucle hacer
    si VACIA?(FRONTERA) entonces devolver fallo
    nodo ← POP(FRONTERA) /* elige el nodo con menor f */
    si problema.TEST-META(nodo.ESTADO) entonces
      devuelve Solucion(nodo)

    añadir nodo.ESTADO a EXPLORADOS

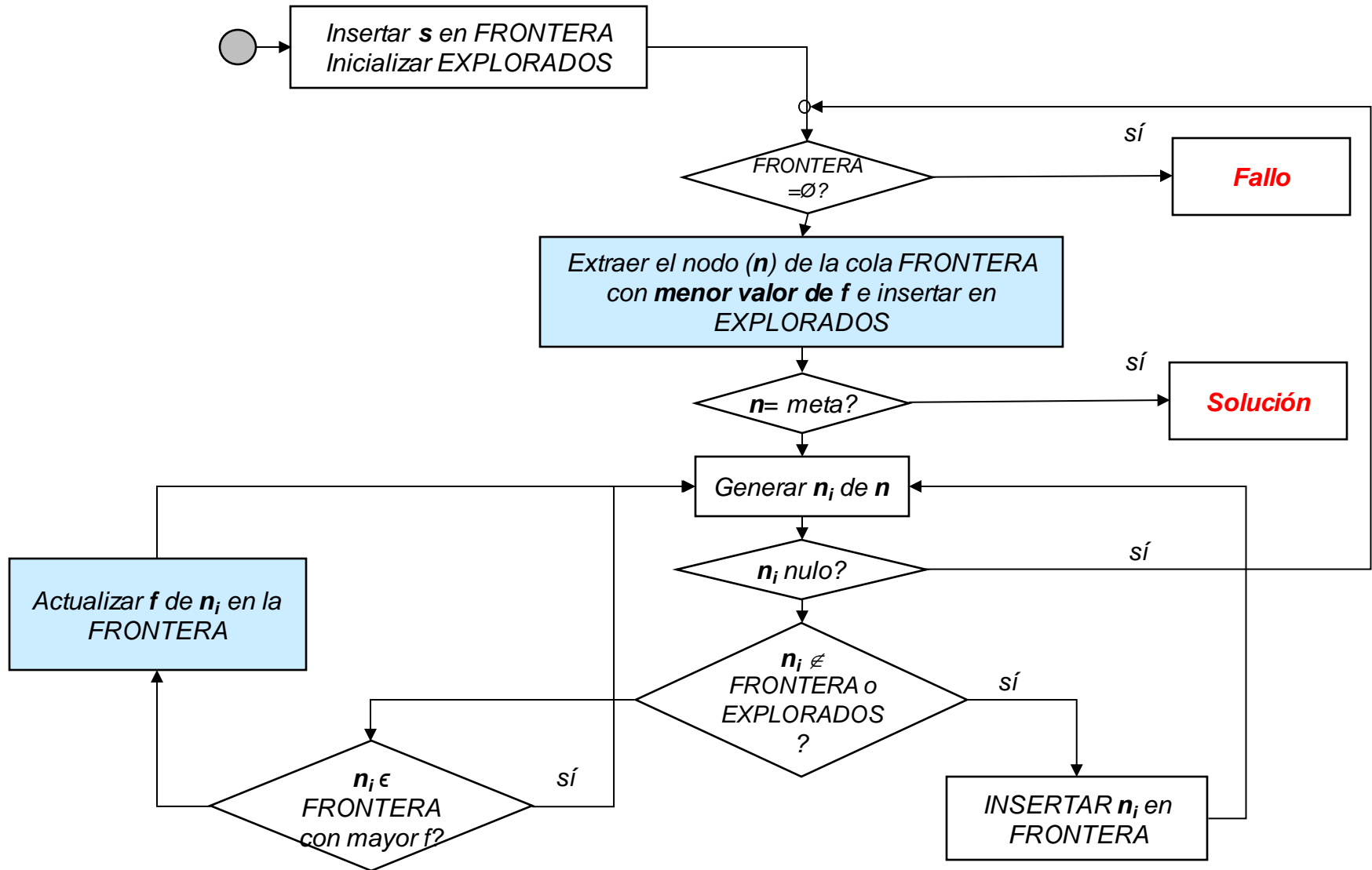
    para cada accion en problema.ACCIONES(nodo.ESTADO) hacer
      hijo ← NODO-HIJO(problema, nodo, accion)
      si hijo.ESTADO no esta en EXPLORADOS o FRONTERA entonces
        FRONTERA ← Insertar(hijo, frontera)
      sino si hijo.ESTADO esta en FRONTERA con menor F-EVALUACION
        entonces reemplazar ese nodo con hijo
```

Si el *nodo* ya se ha explorado, no se actualiza su valor de *f*

```
funcion NODO-HIJO (problema, padre, acción) devuelve un nodo
  devuelve un nodo con
    ESTADO = problema.RESULTADO(padre.ESTADO, accion)
    PADRE = padre, ACCION = accion
    FUNCIÓN-EVALUACIÓN = padre.COSTE-CAMINO +
      problema.COSTE-PASO(padre.ESTADO, accion) +
      problema.HEURISTICA(nodo.ESTADO)
```

# Estrategias de búsqueda informada

## Búsqueda A\*

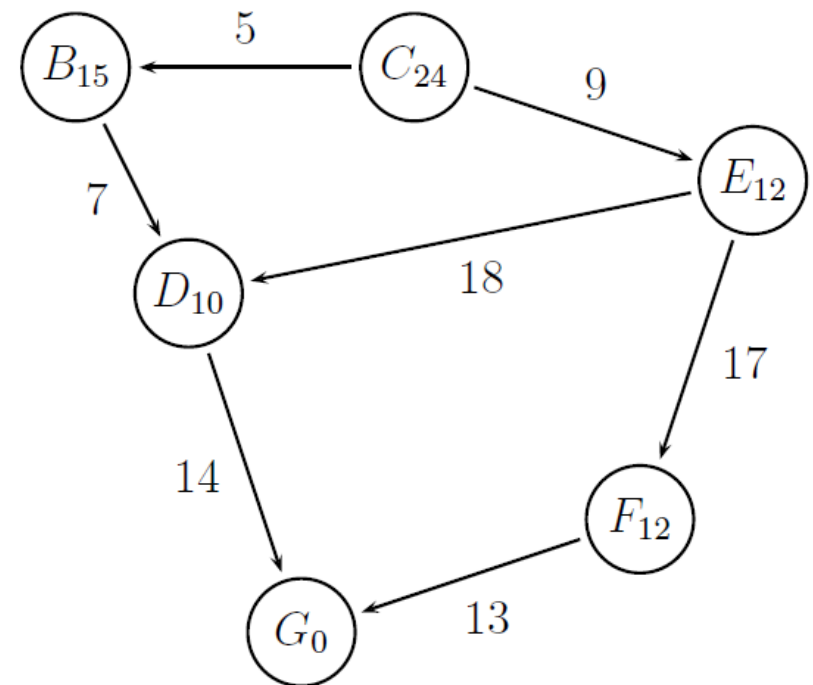


# Estrategias de búsqueda informada

## Ejercicio

- Escribir la secuencia de generación y expansión de nodos con A\*.
- Suponer que la meta es  $G$ .

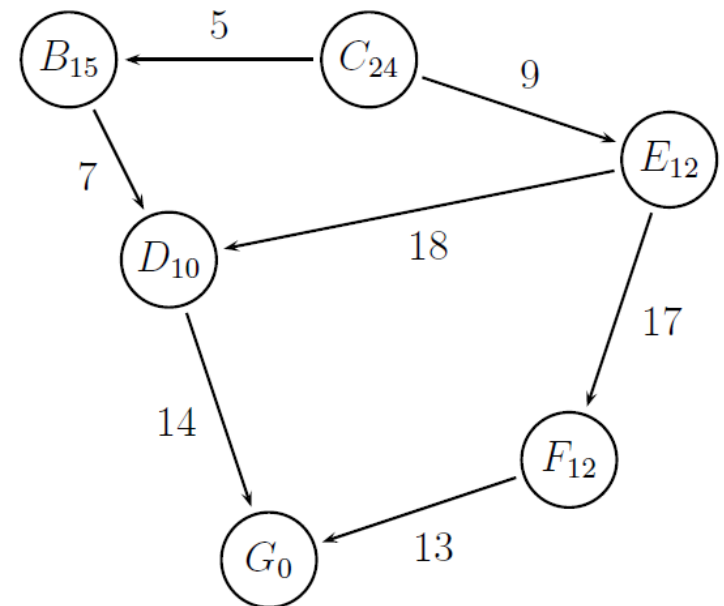
Paso	Frontera	Explorados
1	C (0+24)	-
...	...	...



# Estrategias de búsqueda informada

## Solución

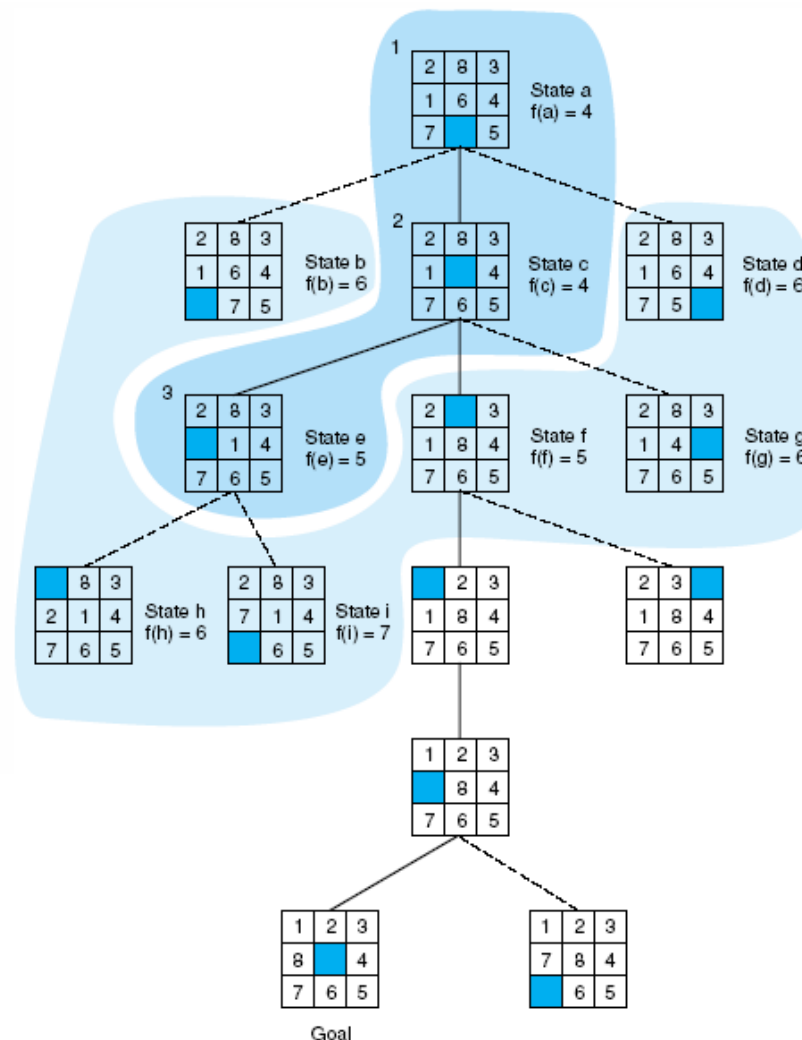
Paso	Frontera	Explorados
1	C(0+24)	-
2	B(5+15),E(9+12)	C(0)
3	E(9+12),D(12+10)	C(0),B(5)
4	D(12+10),F(26+12)	C(0),B(5),E(9)
5	F(26+12),G(26+0)	C(0),B(5),E(9),D(12)
6	F(26+12)	C(0),B(5),E(9),D(12), G(26)



# Estrategias de búsqueda informada

## Búsqueda A\*

Lista de nodos  
*frontera* y  
*expandidos*  
después de la  
tercera  
iteración



*Expandidos*

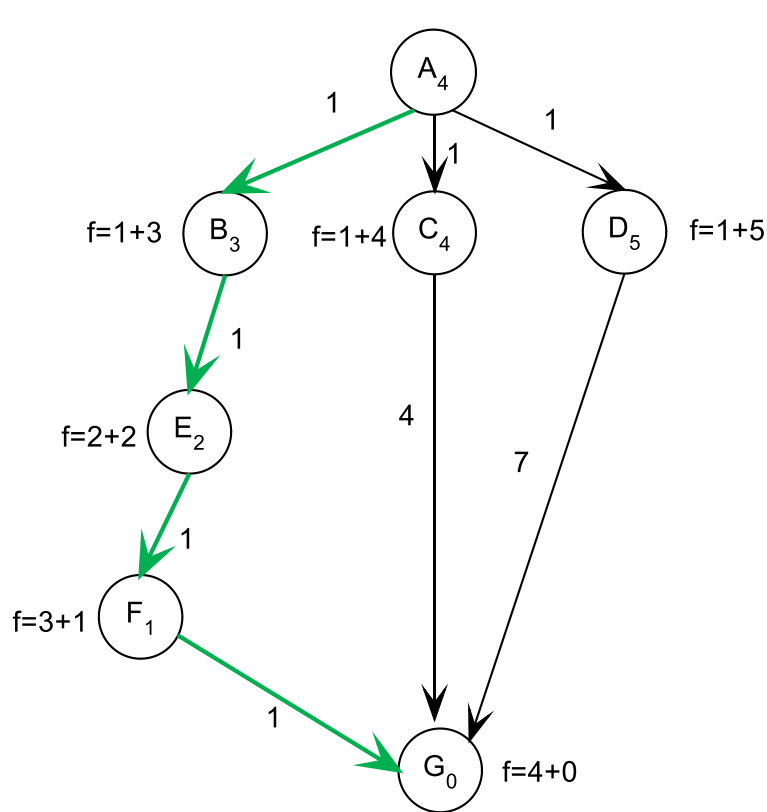
*Frontera*

- La búsqueda A\* basada en árbol es óptima si la heurística es **admisible** (optimista), es decir, si nunca sobreestima el coste real de alcanzar la meta.
- La búsqueda A\* basada en grafo es óptima si la heurística es **consistente** (optimista).
- Si  $g(n)$  es el coste real de alcanzar  $n$  por el camino actual, y  $f(n)=g(n)+h(n)$ , entonces  $f(n)$  nunca sobreestimaré el coste real de una solución por el camino actual que pase por  $n$
- *Problema de rutas en Rumanía*: La distancia en línea recta es una heurística admisible puesto que es la distancia más corta entre dos puntos y nunca podrá sobreestimar

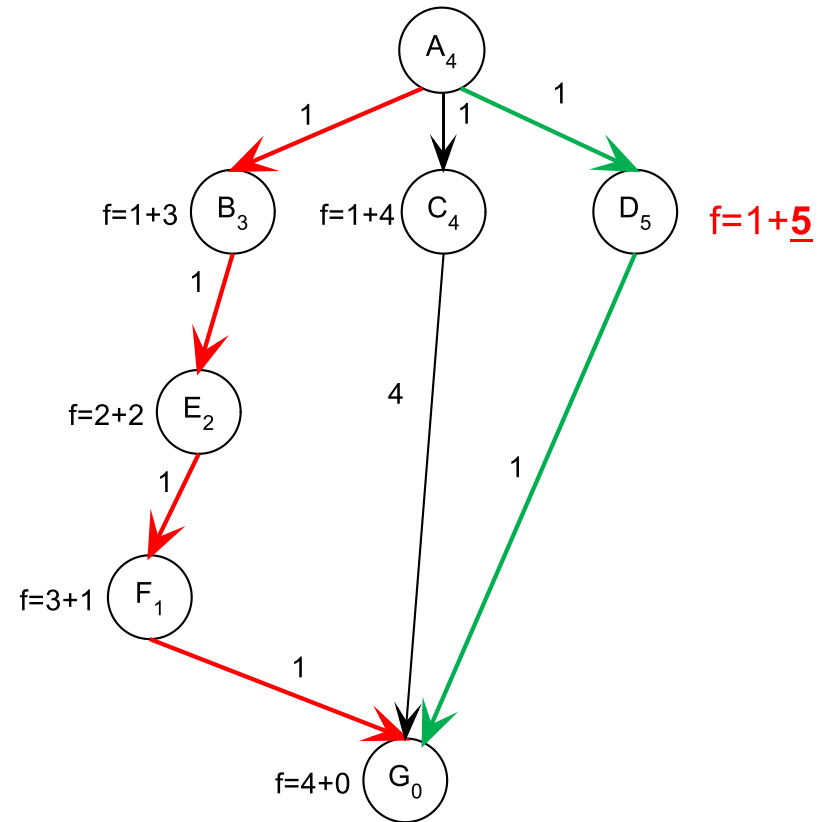


# Estrategias de búsqueda informada

## Efecto de la sobrestimación



$h$  subestima a **coste real**, se elige el camino óptimo



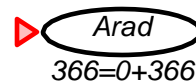
$h$  sobrestima a **coste real**, se elige un camino subóptimo

El subíndice de la etiqueta del nodo corresponde al valor de  $h$  y la etiqueta de los arcos al valor de  $g$

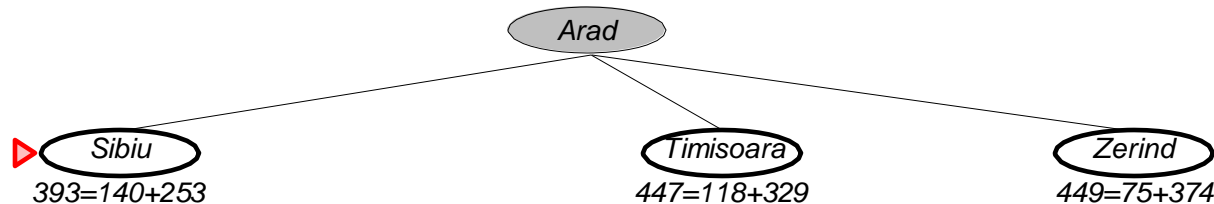
# Estrategias de búsqueda informada

## Búsqueda A\*. El problema de rutas en Rumanía

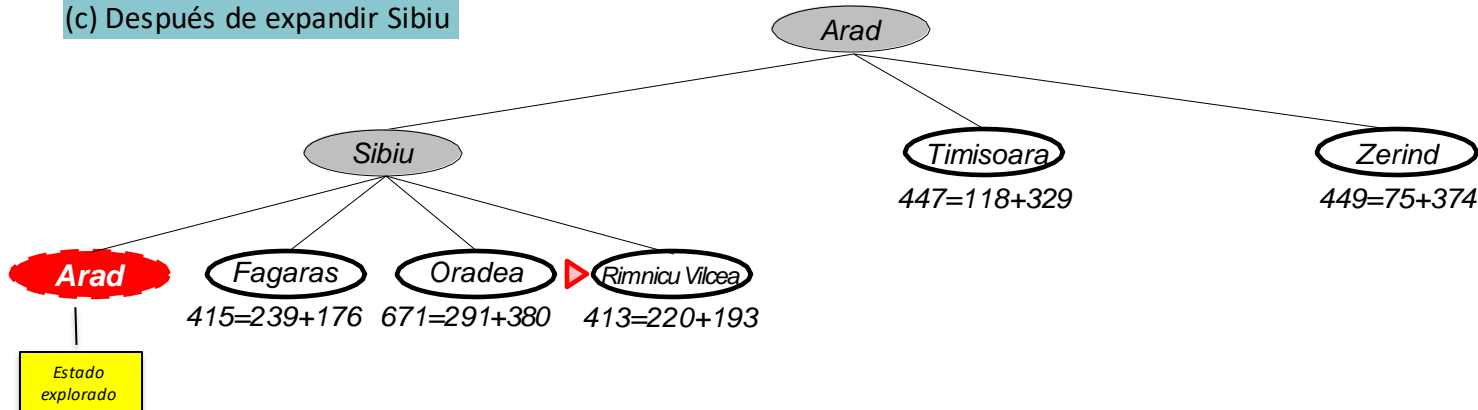
(a) Estado inicial



(b) Después de expandir Arad



(c) Después de expandir Sibiu



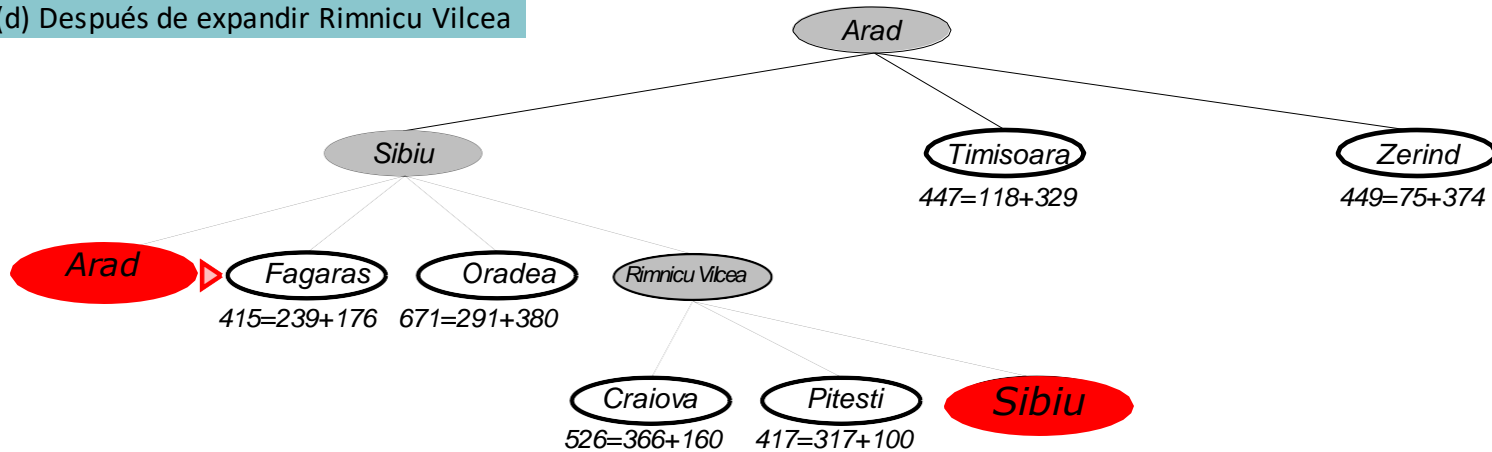
Valores de h  
distancia en línea  
recta a Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

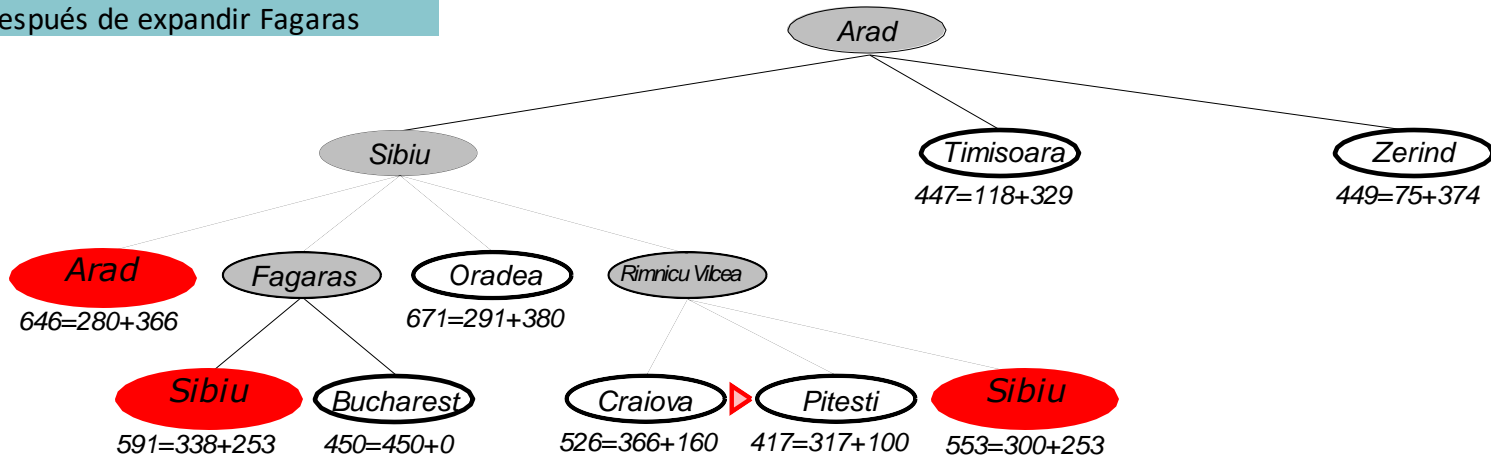
# Estrategias de búsqueda informada

## Búsqueda A\*. El problema de rutas en Rumanía

(d) Después de expandir Rimnicu Vilcea



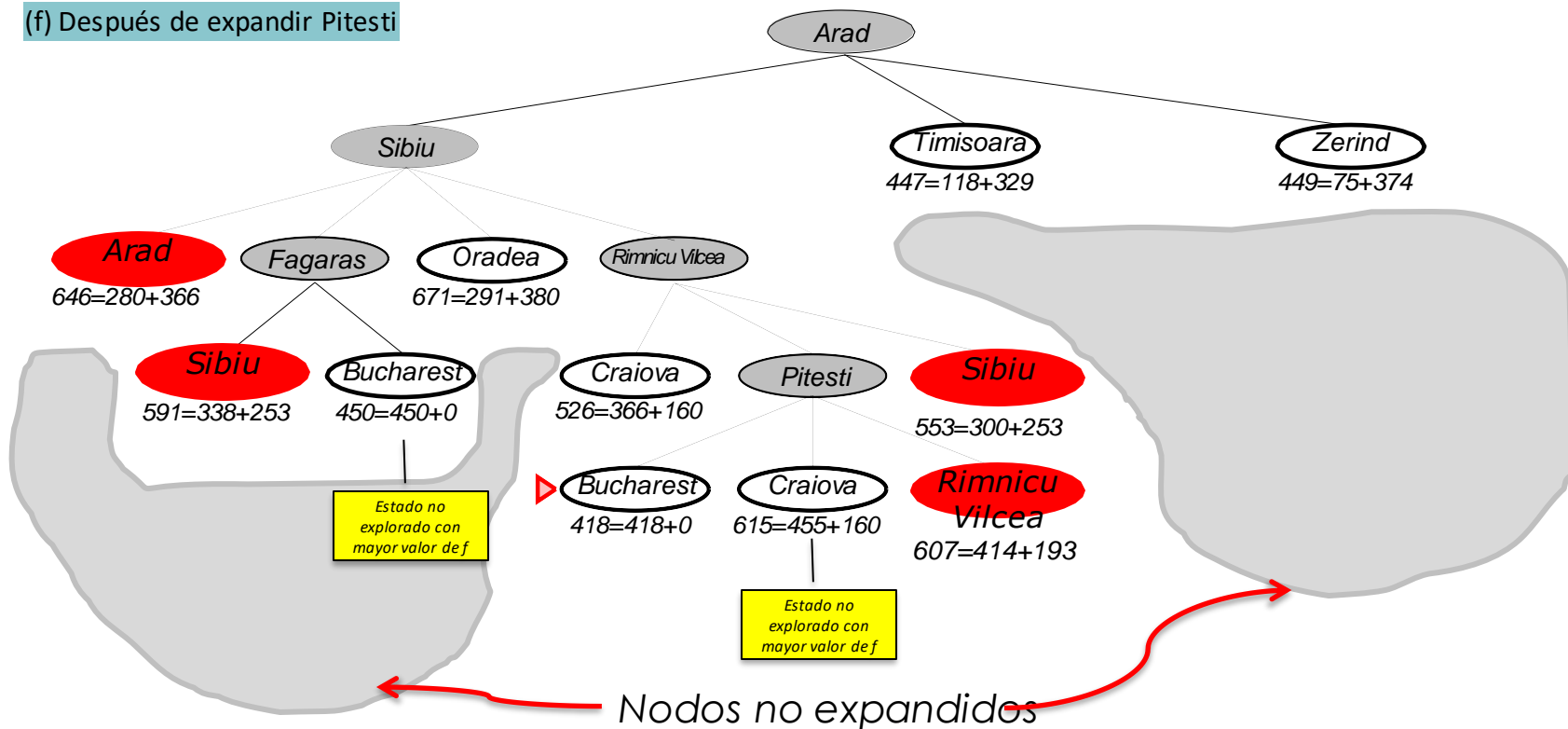
(e) Después de expandir Fagaras



# Estrategias de búsqueda informada

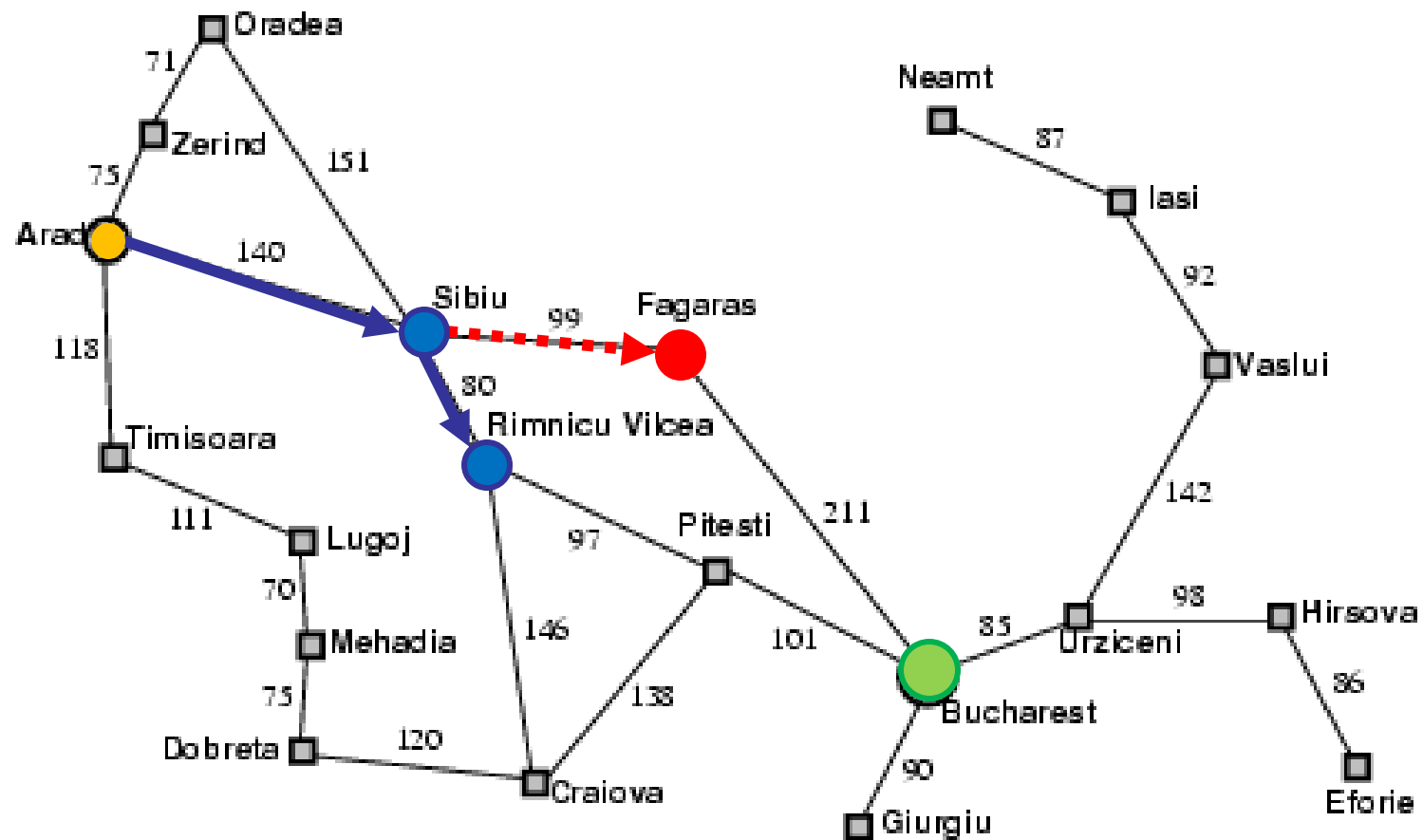
## Búsqueda A\*. El problema de rutas en Rumanía

(f) Después de expandir Pitesti



# Estrategias de búsqueda informada

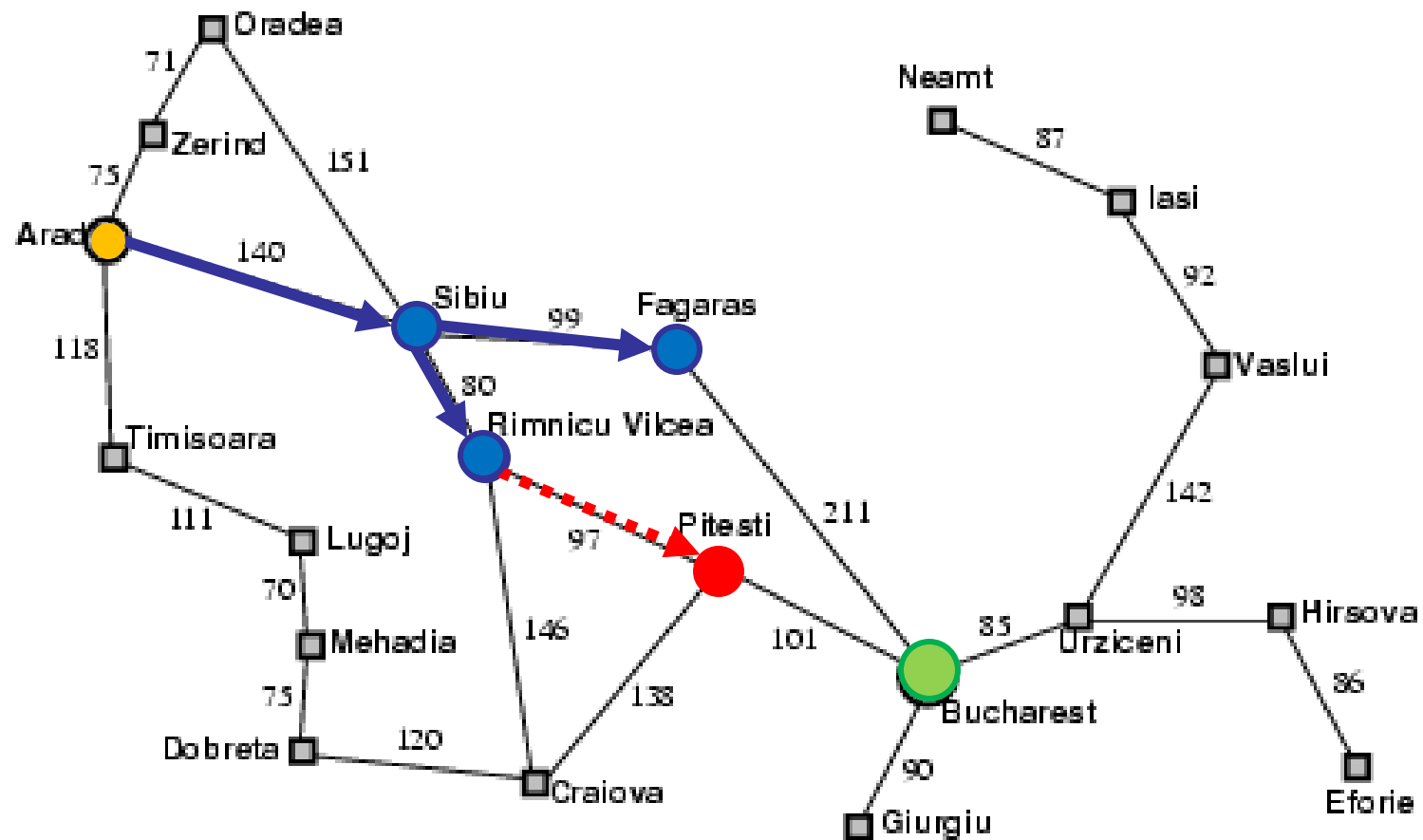
## Búsqueda A\*. El problema de rutas en Rumanía



Se expande **Rimnicu** ( $f=(140+80)+193=413$ ) frente a **Fagaras** ( $f=(140+99)+176=415$ )

# Estrategias de búsqueda informada

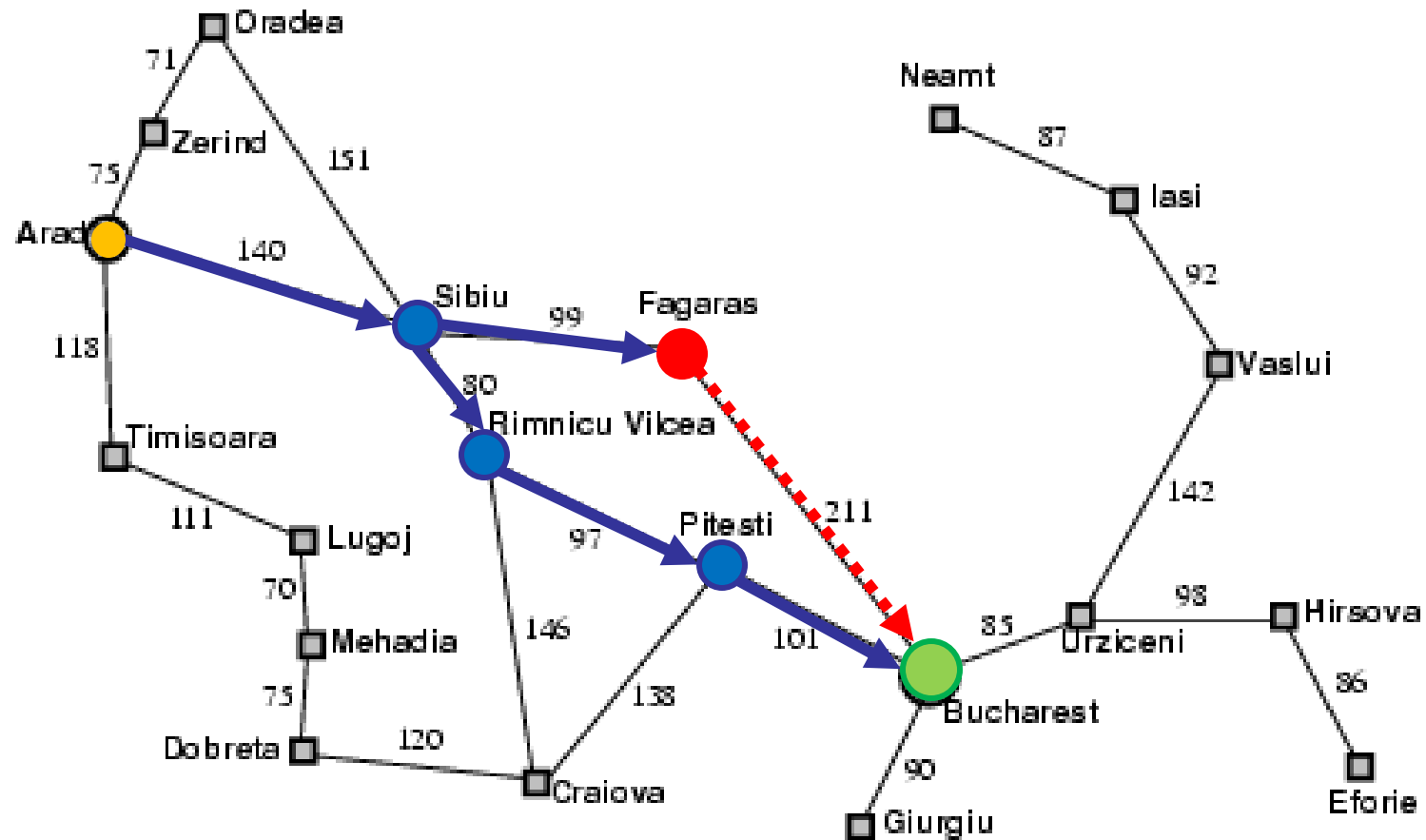
## Búsqueda A\*. El problema de rutas en Rumanía



Se expande **Fagaras** ( $f=(140+99)+176=415$ ) frente a **Pitesti** ( $f=(140+80+97)+100=417$ )

# Estrategias de búsqueda informada

## Búsqueda A\*. El problema de rutas en Rumanía



A través de **Fagaras**  $f(\text{Bucharest}) = 140 + 99 + 211 = 450$

A través de **Pitesti**  $f(\text{Bucharest}) = 140 + 80 + 97 + 101 = 418$

# Búsqueda A\*

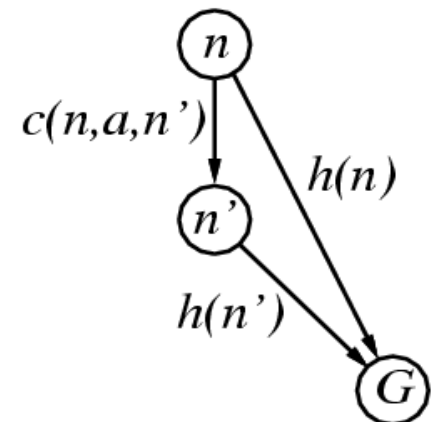
- A\* puede utilizarse para encontrar:
  - un camino de coste total mínimo
  - el camino más rápido posible  $g=1$  (menor nº pasos)
- A\* se comporta como
  - Anchura si:
    - $g$  se incrementa en 1 y  $h = 0$
    - los nodos con igual  $f$  se ordenan de menos a más reciente.
  - Profundidad si:
    - $g = 0$  y  $h = 0$
    - Los nodos se ordenan de más a menos reciente



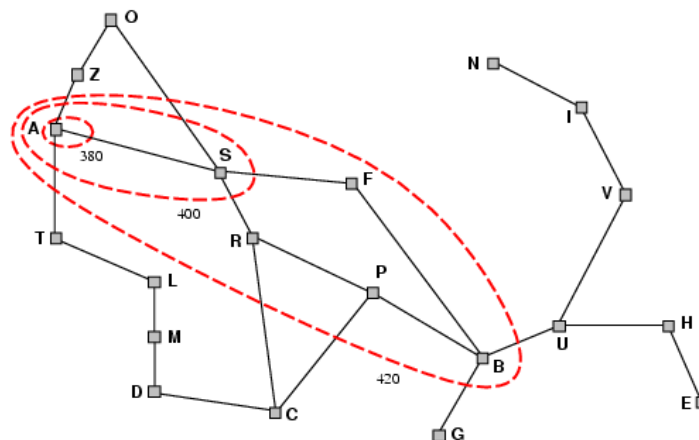
- Cuando existen distintos caminos que llevan a un mismo estado, la búsqueda A\* basada en grafo selecciona siempre el primer camino generado (puede no ser el óptimo). Soluciones:
  - Descartar el camino más costoso de los dos que llegan al mismo nodo.
  - Asegurar que el camino óptimo a cualquier estado repetido es siempre el primero que se encuentra → CONSISTENCIA de una heurística

- La búsqueda A\* basada en grafo es óptima si la heurística es **consistente**:
  - si para cada nodo  $n$  y cada sucesor  $n'$  de  $n$  generado por cualquier acción  $a$ , el coste estimado de alcanzar la meta desde  $n$  ( $h(n)$ ) no es mayor que la suma entre el coste de llegar a  $n'$  y el coste estimado de alcanzar la meta desde  $n'$ .

$$h(n) \leq c(n,a,n') + h(n') \text{ o } h(n) - h(n') \leq c(n,a,n')$$



- Si  $h$  es consistente, entonces los valores de  $f$  a lo largo de cualquier camino son no decrecientes → Monotonía
- El primer nodo meta seleccionado para expansión deberá ser una solución óptima, ya que todos los nodos posteriores serán, al menos, tan costosos.



- Óptima:
  - Ningún otro algoritmo óptimo garantiza expandir menos nodos
  - A\* es óptimamente eficiente para una heurística consistente dada
- Completa
  - Si todos los pasos exceden un  $\epsilon$  finito y b finito
- Complejidad temporal: Exponencial
- Complejidad espacial: Mantiene todos los nodos generados en memoria.
  - No es práctico para problemas muy grandes.

# Búsqueda $A^*$

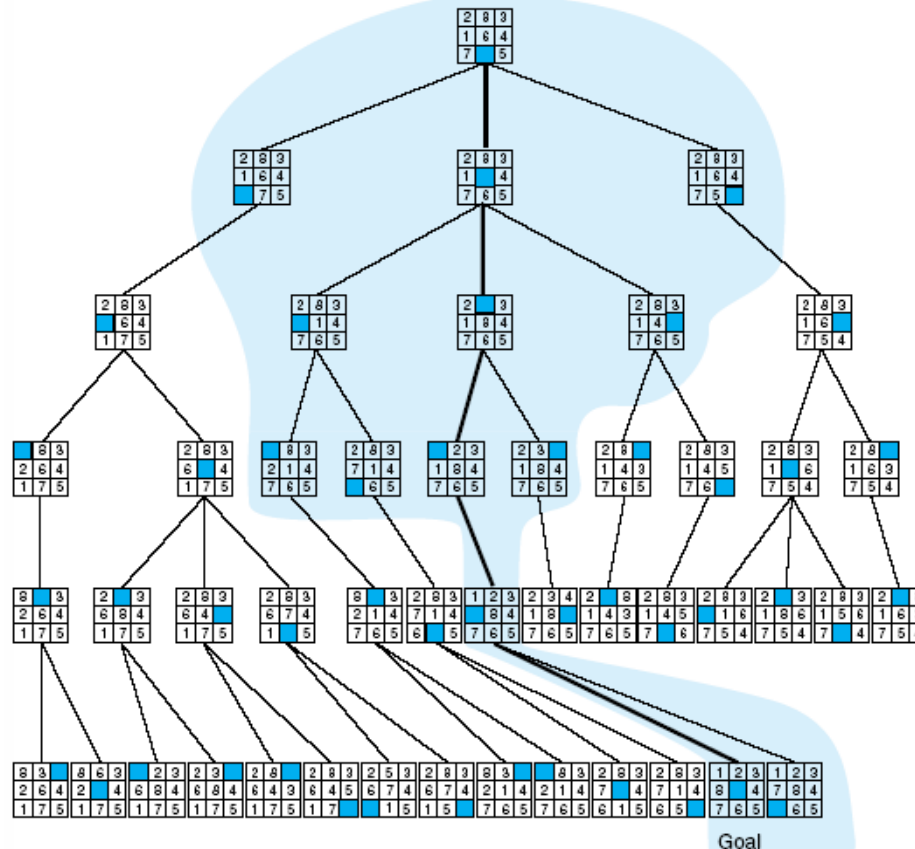
- El tiempo computacional de  $A^*$  no es su desventaja principal.
  - Mantiene todos los nodos generados en memoria
    - Se queda sin espacio antes de quedarse sin tiempo.
  - $A^*$  no es práctico para problemas muy grandes.
- Otros algoritmos de menor coste espacial y temporal son:
  - Búsqueda heurística con memoria acotada.

## Búsqueda A\*

- Usar...
  - Se desea obtener la mejor solución
  - Existe información de tipo heurístico disponible
- Evitar usar...
  - No existen buenas heurísticas disponibles

# Estrategias de búsqueda informada

## Comparativa A\* vs preferente en anchura



Espacio de estados generado usando A\* frente anchura. La porción del grafo recorrido heurísticamente aparece sombreado. El camino óptimo se ha dibujado en negrita. La heurística usada tiene en cuenta el número de piezas descolocadas.

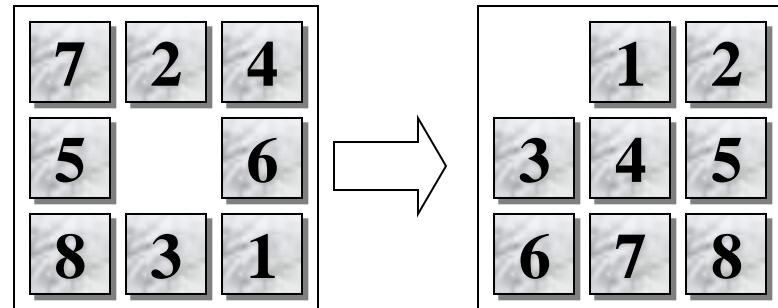
# Estrategias de búsqueda informada

## Heurísticas

- Operadores: Mover una pieza  $\uparrow \downarrow \leftarrow \rightarrow$
- Coste de una solución = número de movimientos necesarios para llegar del estado de partida al estado meta

- Solución típica:

- Consta de 22 pasos
- El factor de ramificación es 3
- La búsqueda exhaustiva en árbol examinaría  $3^{22} \approx 3,1 \times 10^{10}$  estados
- La búsqueda en grafo sólo permite alcanzar 181.440 estados distintos
- Para el puzle 15 serían  $10^{13}$
- Es aconsejable buscar una función heurística para encontrar el camino más corto usando A\*





- Heurísticas basadas en la abstracción del problema:
  - La solución a un problema se construye aplicando una serie de operadores elementales sujetos a una serie de restricciones
  - Conseguir una abstracción de un problema transformándolo en otro más sencillo eliminando ciertas restricciones del problema original (“relajándolo”)
  - La función de coste de la ruta en el problema relajado será una *heurística admisible y consistente* en el problema original
  - Cuantas más restricciones tengamos en cuenta, más precisa será la heurística

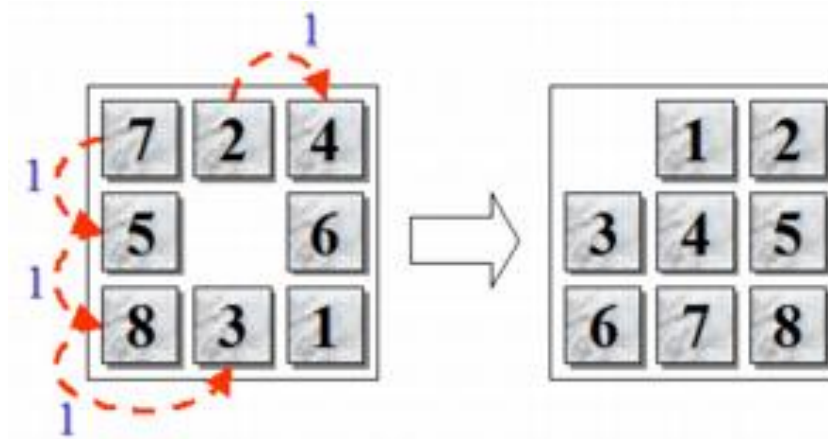
## 8 puzzle. Relajación del problema

- Operador: Mover una pieza de A a B
- Restricciones:
  - si A es adyacente (vertical u horizontal) a B y
  - B está vacío
- Problemas relajados:
  - a) Se puede mover una pieza de A a B, si A es adyacente a B (solapamiento)
  - b) Se puede mover una pieza de A a B, si B está vacío (teletransporte)
  - c) Se puede mover una pieza de A a B (solapamiento y teletransporte)

# Estrategias de búsqueda informada

## Heurística distancia Manhattan

- Caso a) Se puede mover una pieza de A a B, si A es adyacente a B



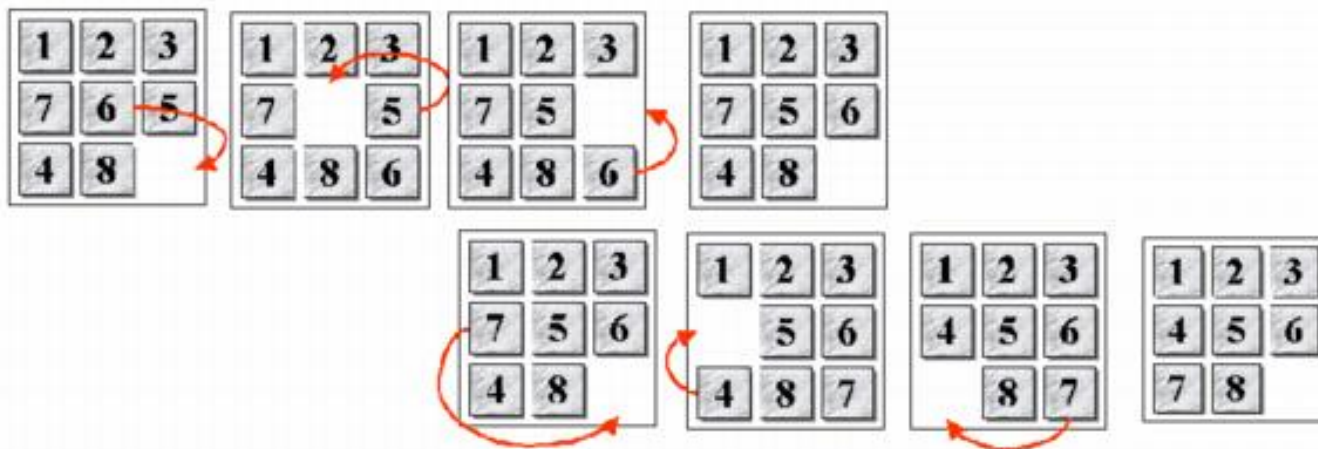
$$H_1 = \text{Movimientos}(\boxed{7}) + \text{Movimientos}(\boxed{2}) + \dots = 3 + 1 + \dots = 18$$

Coste de la solución óptima = 26

# Estrategias de búsqueda informada

## Heurística distancia Gaschnig

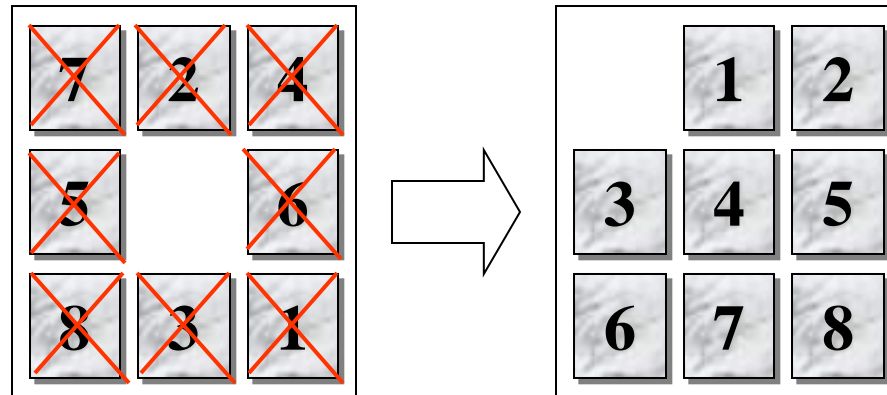
- Caso b) Se puede mover una pieza de A a B, si B está vacío



$H_2$  = número mínimo de movimientos necesarios = 6

Coste de la solución óptima = 18

- Caso c) Se puede mover una pieza de A a B



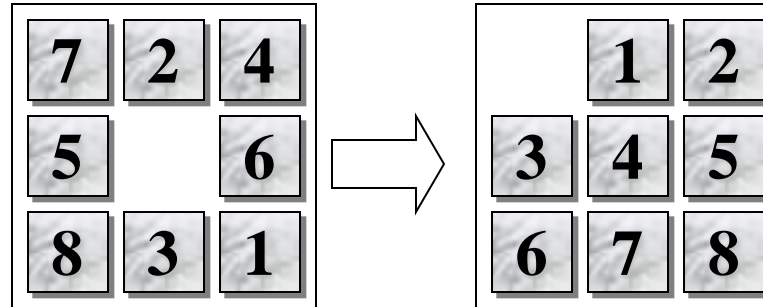
Mal colocada

$$H_3 = 8$$

Coste de la solución óptima = 26

# Estrategias de búsqueda informada

## Heurísticas



- Sean:
  - $h_1$  = número de piezas mal colocadas.
    - Admisible, cualquier pieza mal colocada debe moverse por lo menos una vez
  - $h_2$  = suma de las distancias de las piezas a su posición meta.
    - Admisible, cualquier movimiento supone acercar una pieza un paso más a la meta
- Si una heurística domina a otra ( $\forall n, h_2(n) \geq h_1(n)$ ),  $A^*$  usando  $h_2$  expandirá menos nodos que usando  $h_1$



$h_2(n)$  está más informada que  $h_1(n)$

# Diseño de heurísticas admisibles

- Las heurísticas devuelven la solución precisa del camino en versiones *simplificadas* del problema
  - Se puede mover una pieza a cualquier hueco,  $h_1$
  - Se puede mover una pieza en cualquier dirección,  $h_2$
- Problema *relajado*:
  - el número de restricciones sobre las acciones disponibles es menor
  - el grafo del espacio de estados es un supergrafo del grafo del problema original: la relajación *añade* aristas
- Cualquier solución óptima en el problema original es una solución en el problema relajado...y ¿al revés?
- Conclusiones:
  - Ya que hay caminos más cortos (atajos) en el problema relajado (hay más aristas), el coste de una solución óptima a un problema relajado es una heurística *admisible* para el problema original.
  - Ya que la heurística derivada es el coste exacto en el problema relajado, cumplirá la desigualdad triangular y será *consistente*

# Búsqueda conducida por agendas

- Listas ordenadas de tareas que puede realizar un sistema
- Cada tarea lleva asociada:
  - una lista de razones o motivos por los que dicha tarea es interesante (justificaciones)
  - un valor que representa el peso total de la evidencia que sugiere que la tarea es útil
- No todas las justificaciones de una misma tarea tienen por qué pesar lo mismo
- La búsqueda conducida por agendas es un método de búsqueda por el mejor nodo en el que:
  - Se debe elegir la mejor tarea de la agenda
  - Se debe ejecutar la tarea seleccionada asignando para ello recursos. Si se crean subtareas se insertan en su lugar correspondiente



# Búsqueda conducida por agendas

## Función Búsqueda–agenda

**hacer** mientras no se alcance un estado objetivo o la agenda este vacía

Identificar la mejor tarea de la agenda y ejecutarla

**Si** se han generado nuevas tareas (subtareas) **para cada** una de ellas

**hacer**

**Si** no estaba ya en la agenda **añadir** la nueva tarea

**Si** estaba ya en la agenda

**hacer**

**Si** no tiene la misma justificación **añadir** la nueva justificación

**Si** se ha añadido una tarea o una justificación

calcular el peso asignado combinando la evidencia de todas sus justificaciones  
reorganizar la agenda

## Lo que no debes olvidar

- Estrategias de búsqueda no informada:
  - Anchura y Profundidad. Criterios de calidad. Cuándo usar y cuándo evitar el uso.
- Estrategias de búsqueda informada: A\*.
- Concepto de heurística. Admisibilidad y consistencia.
- Diseño de heurísticas.

## Estrategias de búsqueda informada (búsqueda local)



# Búsqueda local

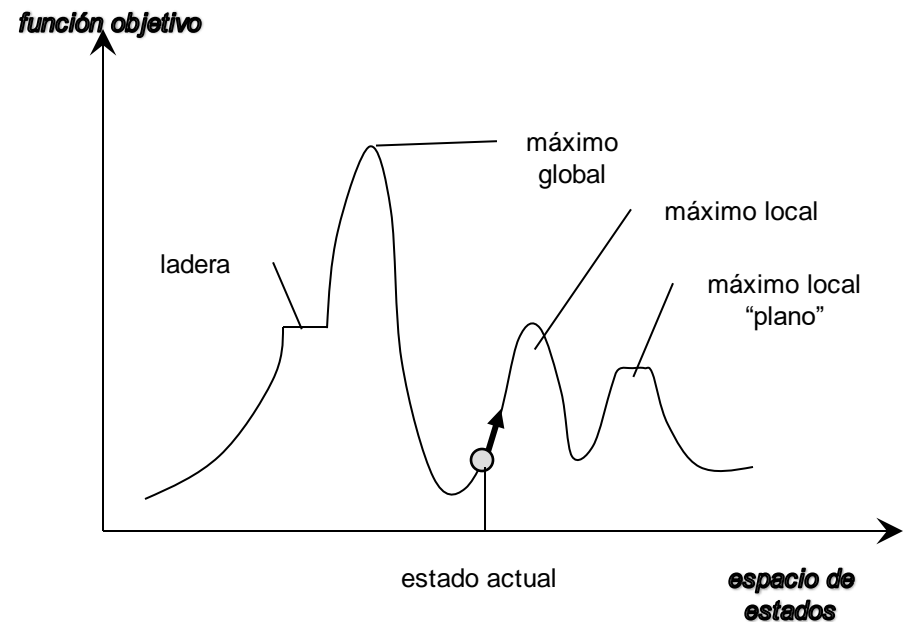
- Búsqueda no informada y búsqueda por el mejor:
  - Exploran sistemáticamente el espacio de búsqueda conservando uno o más caminos en memoria
  - Registran qué alternativas han sido exploradas en cada punto de ese camino y cuáles no
  - Cuando encuentran una meta, el camino a la meta constituye una solución al problema.
- En muchos problemas, el camino es irrelevante:
  - 8 reinas
  - Diseño de circuitos integrados
  - Programación del trabajo en una empresa
  - Optimización de redes de telecomunicaciones
  - .....

# Búsqueda local

- Usan un único estado actual (en lugar de múltiples caminos) y se mueven sólo a sus vecinos (el camino a la meta es irrelevante).
- Los caminos no se retienen.
- No son sistemáticos, pero tienen dos ventajas:
  - Poca memoria, por lo general una cantidad constante.
  - Encuentran soluciones razonables en espacios de estados grandes o infinitos (continuos),
  - Aptos para problemas de optimización
    - Encontrar el mejor estado de acuerdo con una función objetivo.

# Algoritmo de escalada (“Hill-climbing”)

- Cada punto representa un estado y un valor de la función objetivo
- Se trata de encontrar el punto más alto, máximo global\*

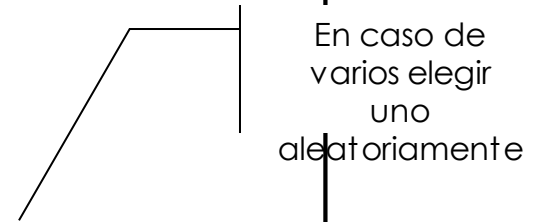


\* A efectos prácticos, consideraremos que el algoritmo intenta maximizar una función objetivo; en el caso de función heurística, en realidad intenta minimizar la función heurística

# Algoritmo de escalada\*

```
función HILL-CLIMBING (problema) devuelve un estado que es un máximo local
entradas: un problema
variables locales : actual, un nodo
                  vecino, un nodo

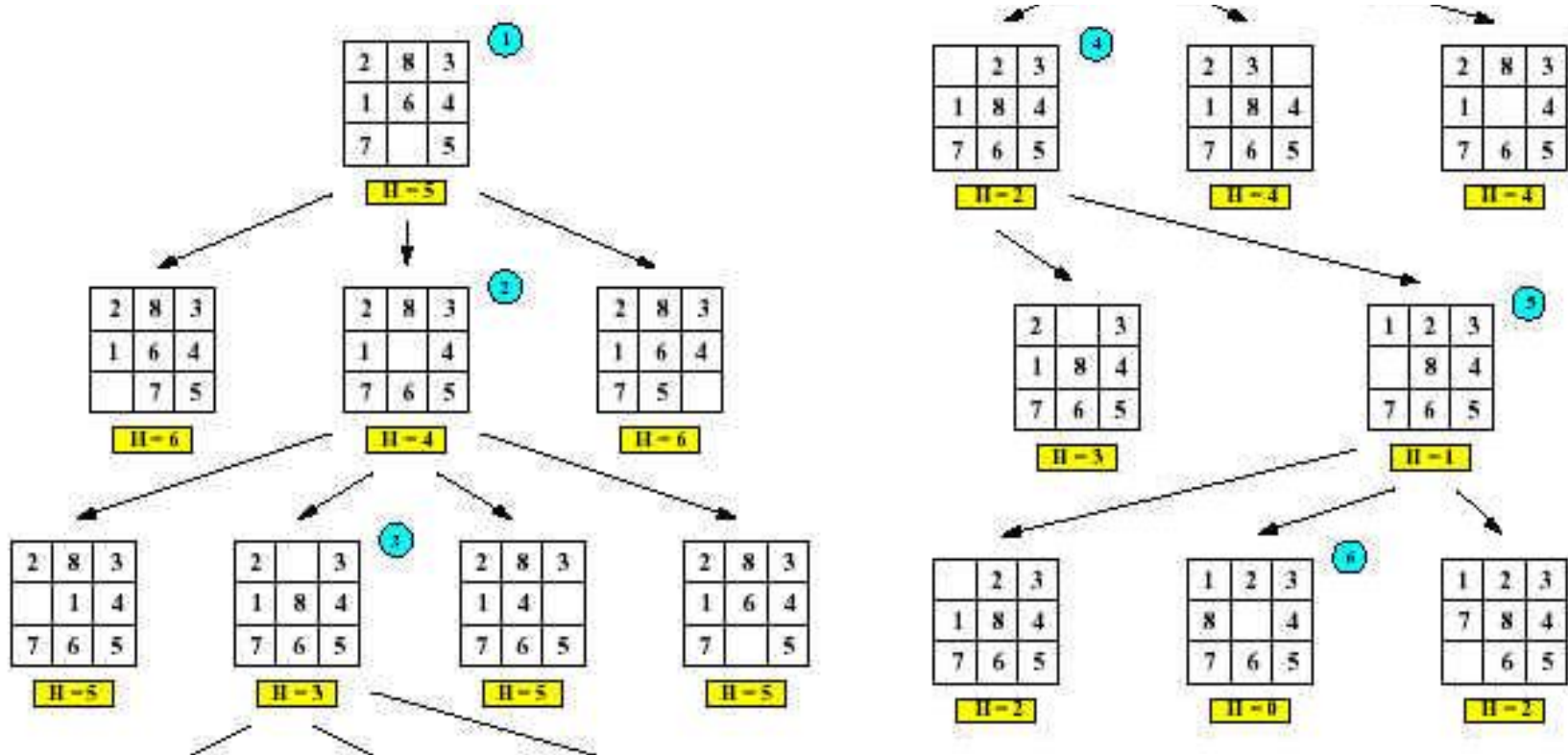
actual ← HACER_NODO(ESTADO_INICIAL[problema])
bucle hacer
    vecino ← sucesor de actual con mejor valor de función
    si VALOR[vecino] ≤ VALOR[actual]
    entonces devolver ESTADO[actual]
    actual ← vecino
```



- Se mueve continuamente en la dirección incremental del valor y termina cuando ningún descendiente tiene mejores valores.
- No mantiene un árbol de búsqueda, sólo el nodo actual (estado y valor de la función objetivo).

\* Esta es la versión "steepest ascent" o de máxima pendiente; en el algoritmo clásico se selecciona el primer sucesor mejor que el estado actual, es decir,  $VALOR[vecino] < VALOR[actual]$ .

# Algoritmo de escalada: Ejemplo









$h$  = Suma de las distancias Manhattan

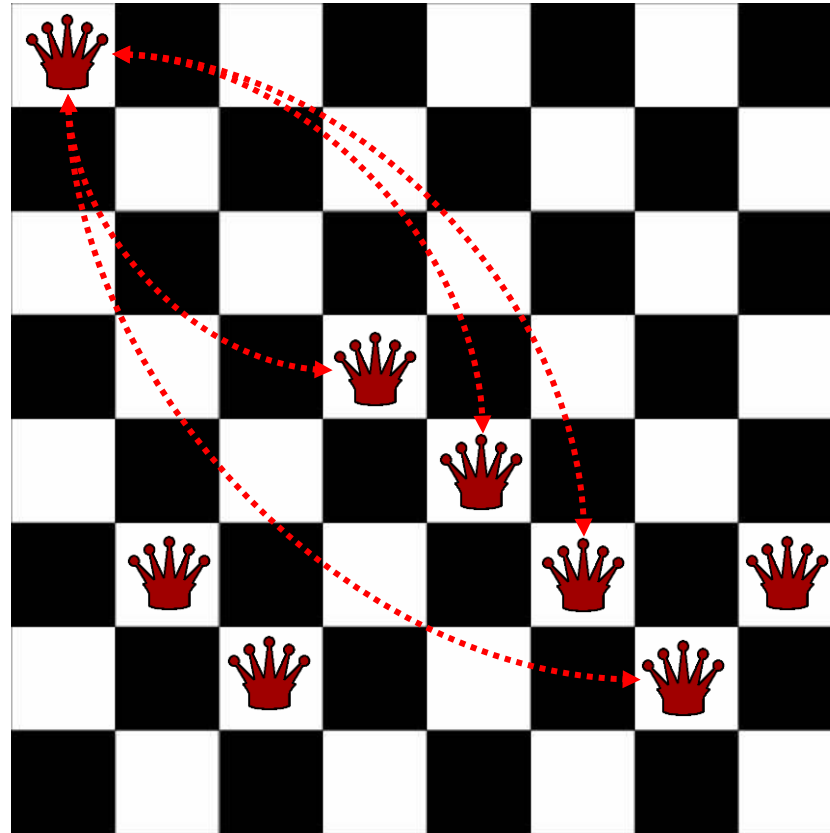


# El problema de las 8 reinas

- Operador: mover una reina dentro de la columna
- Heurística = número de pares de reinas que se atacan
- En esta situación,  $h=12$

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14		13	16	13	16
	14	17	15		14	16	16
17		16	18	15		15	
18	14		15	15	14		16
14	14	13	17	12	14	12	18

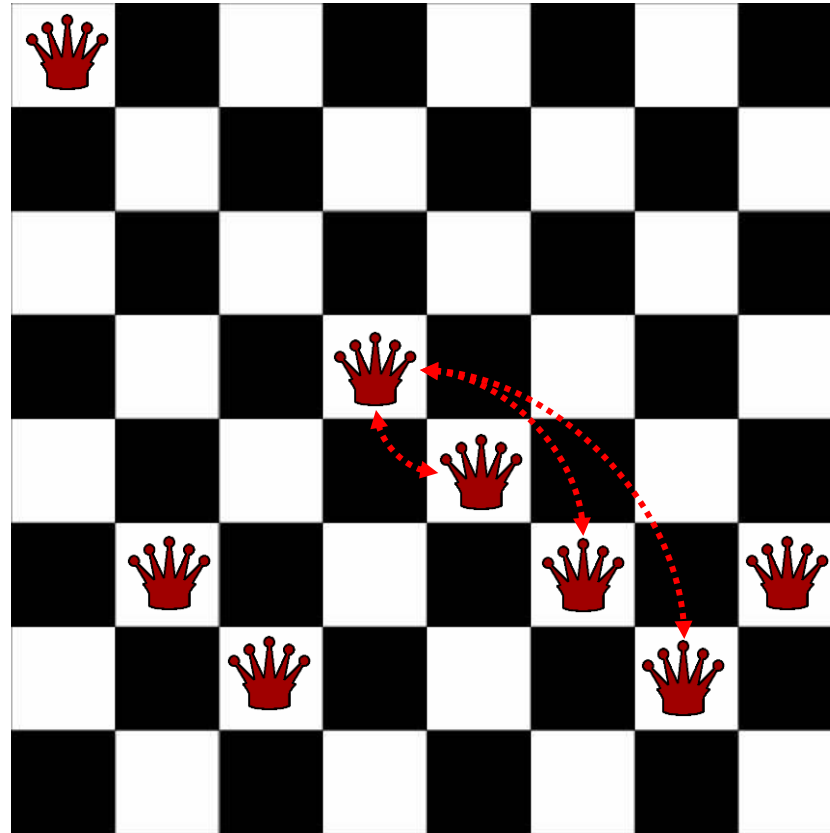
# El problema de las 8 reinas



En esta situación,  $h=18$

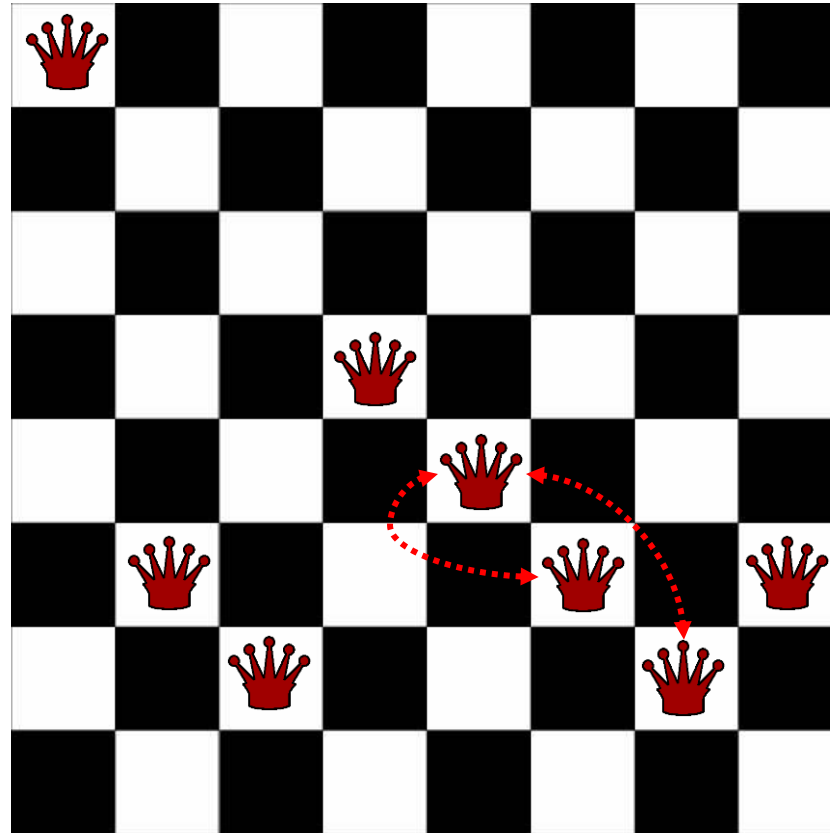
$h$  = número de pares de reinas que se atacan = 4 + ...

# El problema de las 8 reinas



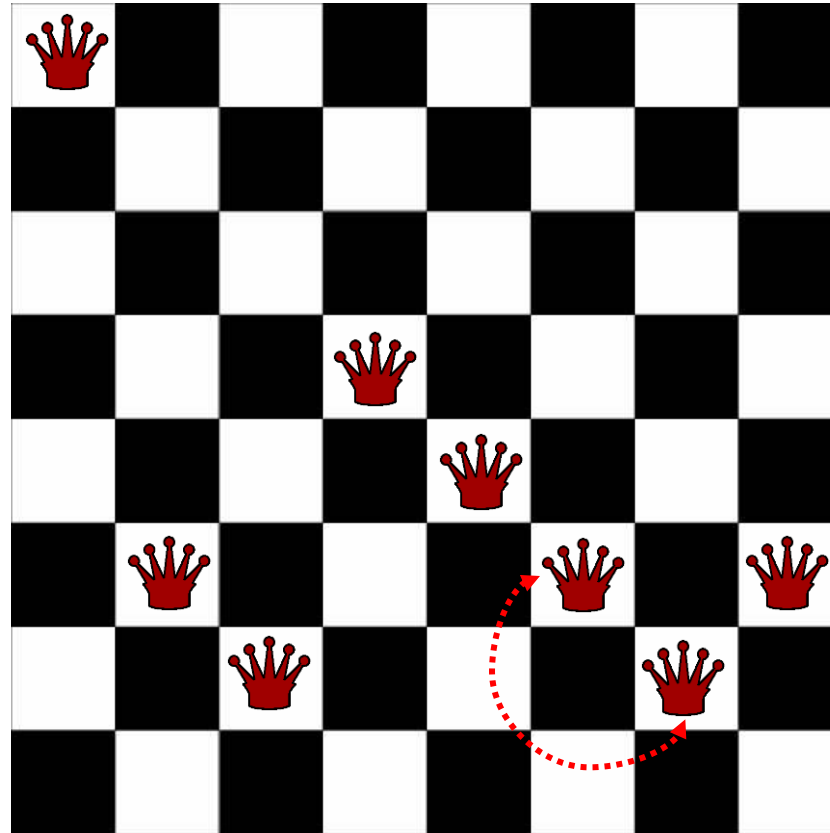
$h = \text{número de pares de reinas que se atacan} = 4 + 3 + \dots$

# El problema de las 8 reinas



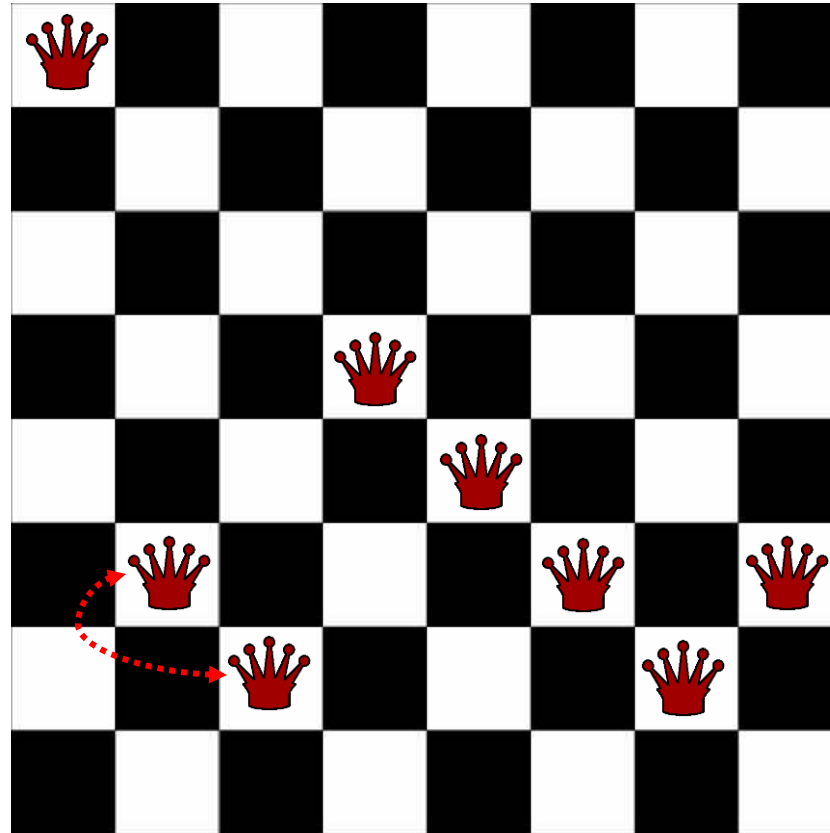
$h = \text{número de pares de reinas que se atacan} = 7 + 2 + \dots$

# El problema de las 8 reinas



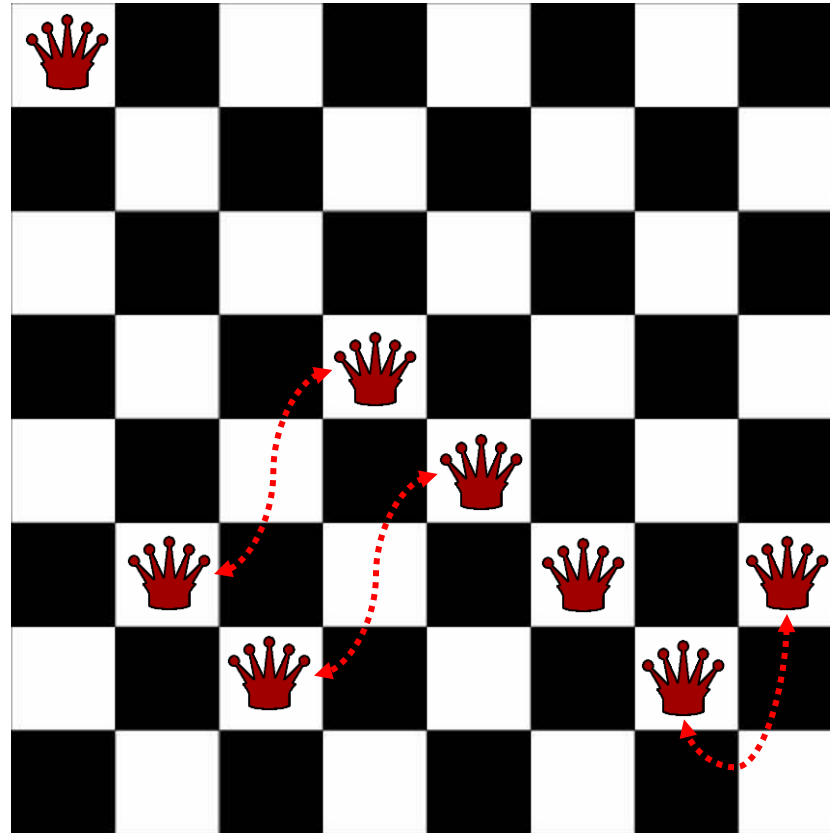
$h = \text{número de pares de reinas que se atacan} = 9 + \mathbf{1} + \dots$

# El problema de las 8 reinas



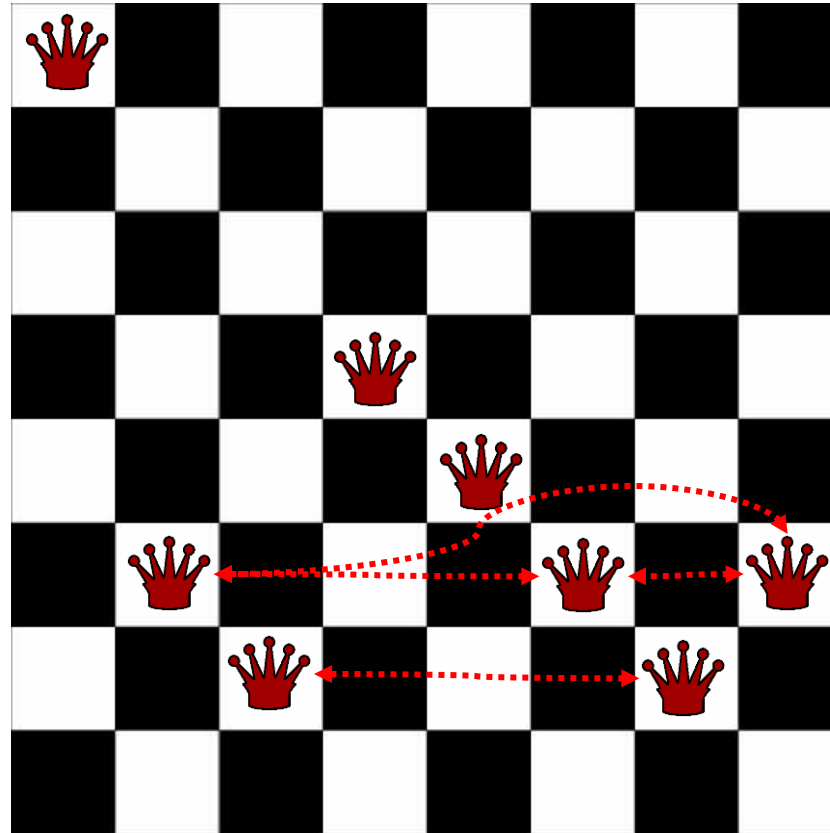
$h = \text{número de pares de reinas que se atacan} = 10 + 1 + \dots$

# El problema de las 8 reinas



$h = \text{número de pares de reinas que se atacan} = 11 + 3 + \dots$

# El problema de las 8 reinas



$h = \text{número de pares de reinas que se atacan} = 14 + 4 = 18$



# Algoritmo de escalada: Inconvenientes

## ■ Máximo Local

- Estado puntual mejor que cualquiera de sus vecinos, pero peor que otros estados más alejados (máximo global). Cuando aparecen cerca de la solución final, se llaman estribaciones

## ■ Meseta

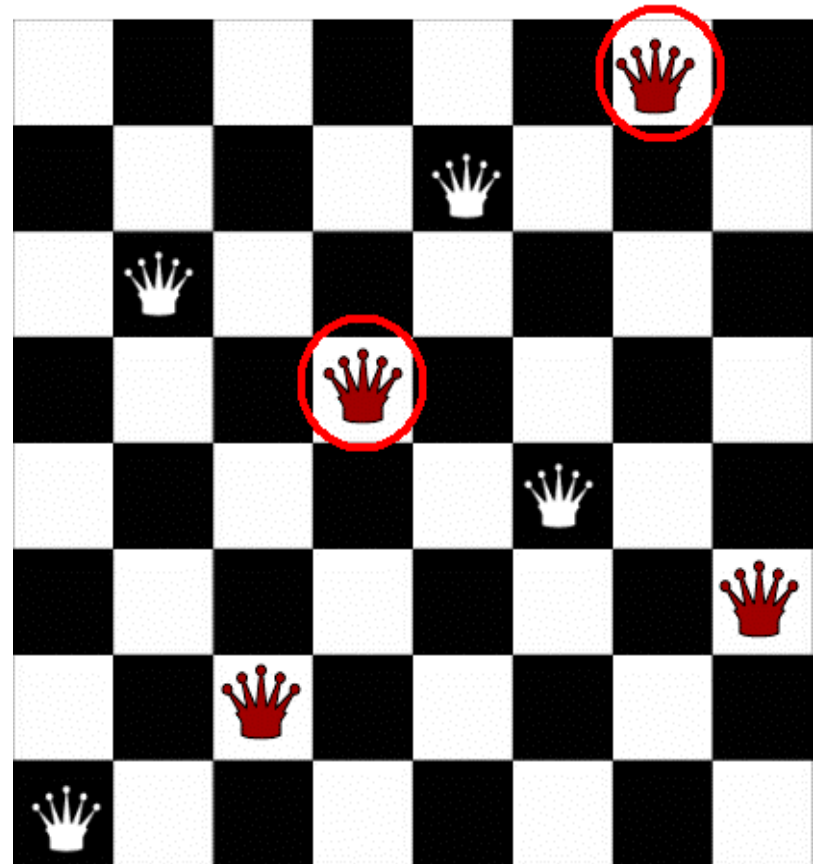
- Región del espacio de estados en la que todos los estados individuales tienen el mismo valor de la función heurística y, por lo tanto, no es posible determinar la mejor dirección para continuar

## ■ Cresta

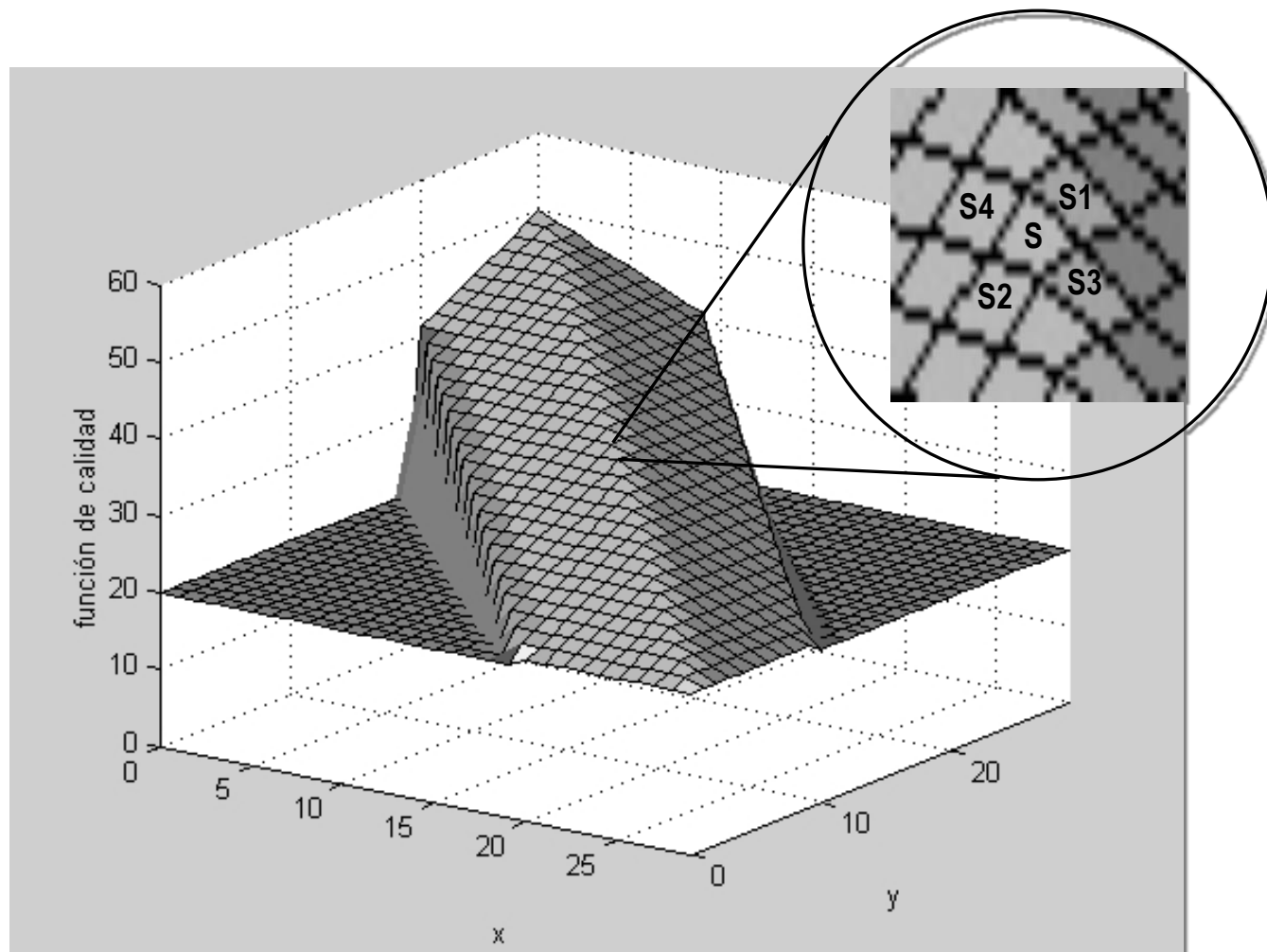
- Región del espacio de estados con estados que tienen mejores valores de la función heurística que los de regiones colindantes, pero a los que no podemos llegar mediante transiciones simples

# Máximo local

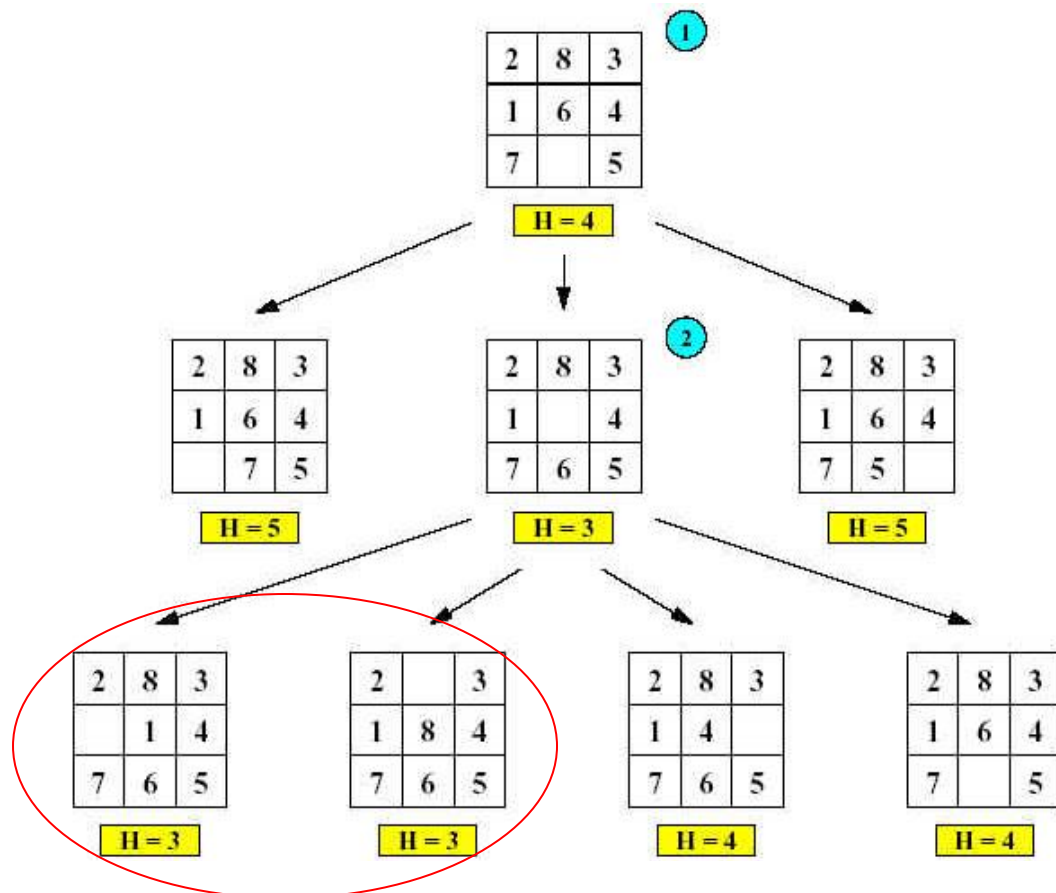
- Operador: mover una reina dentro de la columna
- Heurística = número de pares de reinas que se atacan
- En esta situación,  $h=1$ , y cualquier movimiento empeora este valor



# Crestas



# Mesetas



$h$  = número de piezas mal colocadas

# Algoritmo de escalada: Soluciones

- Para los Máximos Locales
  - Regresar a un nodo previo e intentar una dirección diferente (backtracking)
- Para las Mesetas
  - Realizar un gran salto en el espacio de búsqueda y tratar de alcanzar una región diferente del espacio de estados
- Para las Crestas
  - Aplicar más de un operador antes de realizar la prueba de meta (equivale a moverse en varias direcciones)

# Bibliografía

- Russell & Norvig, *Artificial Intelligence, a modern approach*. Pearson, Prentice Hall, 3rd edition, 2010
- Moret et al., *Fundamentos de Inteligencia Artificial*, Servicio de Publicaciones UDC, 2004
- Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, eds., 1971
- Rich & Knight, *Inteligencia Artificial*, McGraw-Hill, eds., 1994