

Inteligencia Artificial

Resolución de problemas:

Búsqueda informada



Búsqueda informada

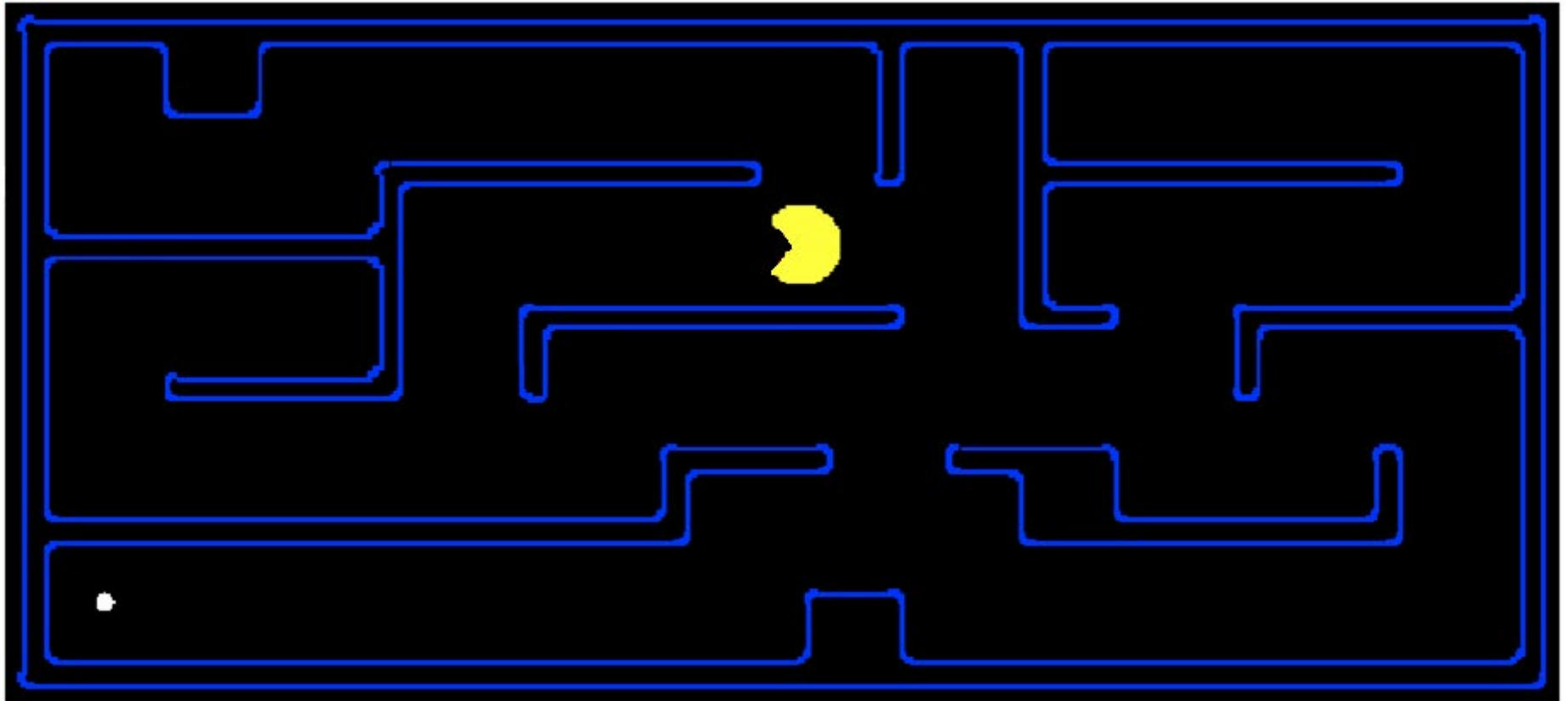
- Heurísticas
- Estrategias de búsqueda informada
 - Búsqueda voraz (greedy) o primero mejor
 - Búsqueda A^*

Heurísticas

- La búsqueda informada mejora a la búsqueda no informada mediante la introducción en el proceso de búsqueda, de **información específica del problema** que permita acelerar la búsqueda.
- Una **heurística** $h(n)$ es una estimación de lo cercano que se encuentra un nodo al objetivo
- Las heurísticas únicamente son válidas para un problema (no son generales).

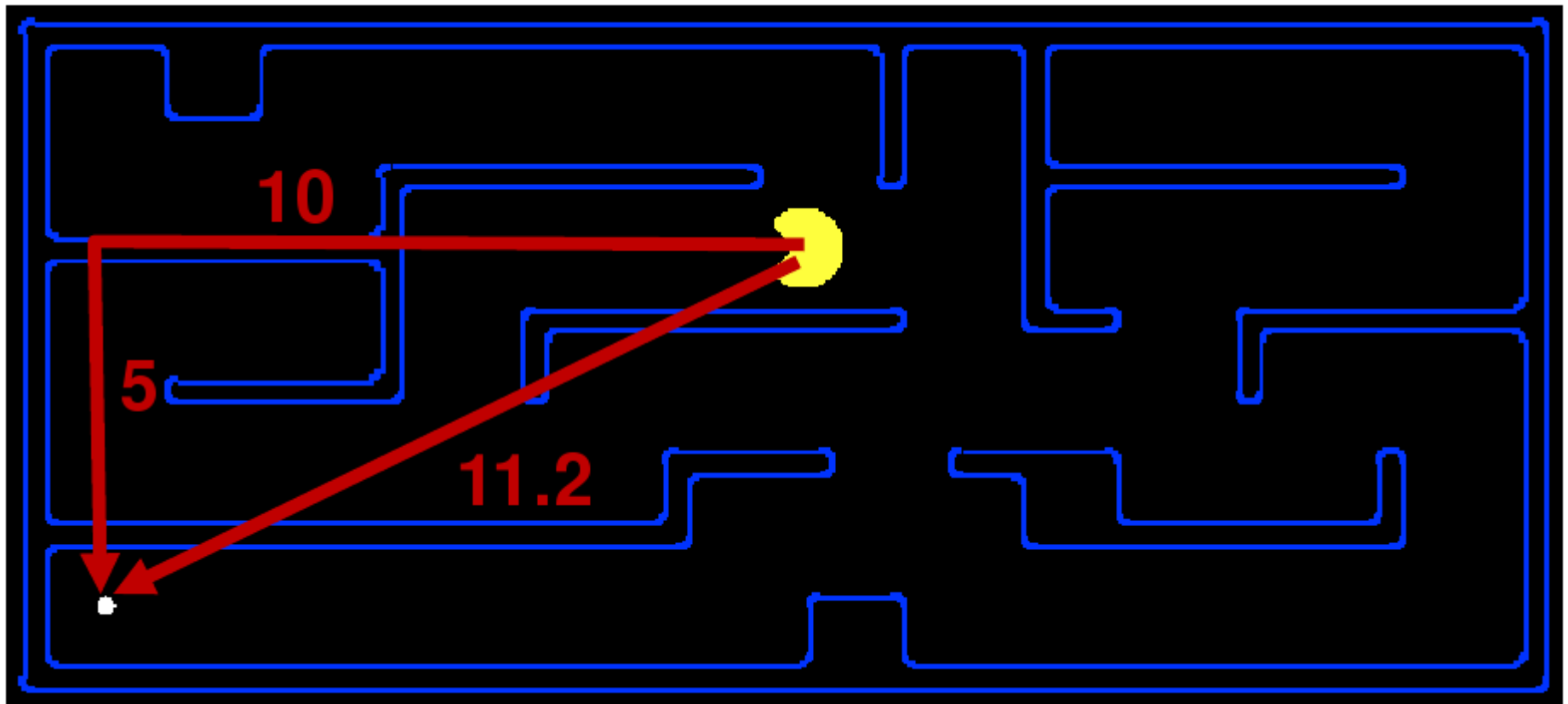
Heurísticas

- Ejemplos en Pacman: ?



Heurísticas

- Ejemplos en Pacman: distancia Euclídea, distancia Manhattan.



Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)

Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Implementación:
 - La *frontera* se ordena siguiendo un orden creciente de la heurística: se sacan primero los nodos de menor heurística

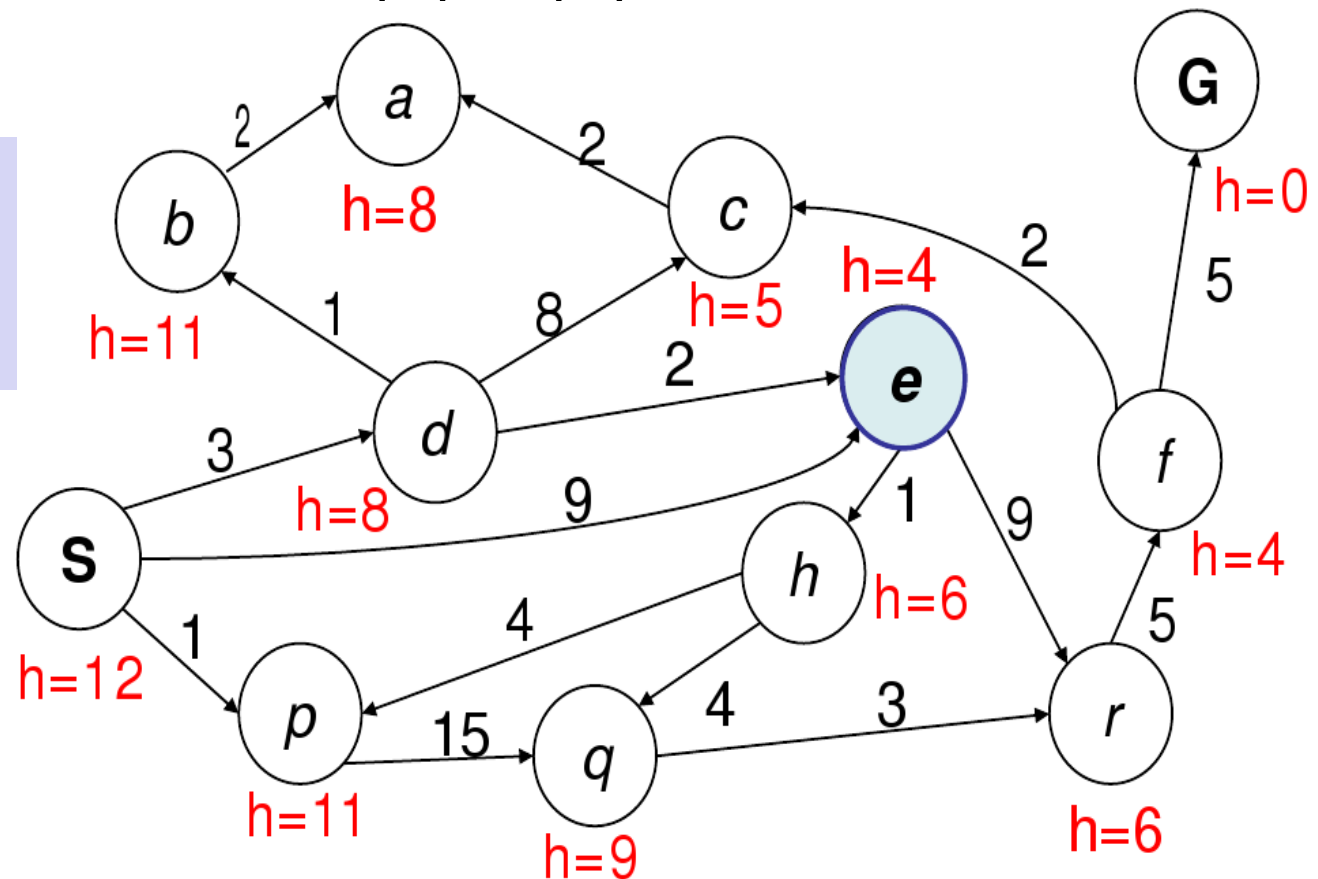
```
función BEST-FIRST-SEARCH(problema, f) devuelve una solución o fallo
  nodo ← CREAR-NODO(problema.estado-inicial), alcanzados ← ∅, frontera ← ∅
  INSERTAR((problema.estado-inicial, nodo), alcanzados)
  INSERTAR(nodo, frontera)
  bucle while not VACIA? (frontera) hacer
    nodo ← SACAR-BORRANDO-PRIMERO(frontera)
    si ES-OBJETIVO (nodo.estado) entonces devolver SOLUCION(nodo)
    bucle for each sucesor en EXPANDIR(nodo, problema) hacer
      s ← sucesor.estado
      si s no está en alcanzados o (sucesor.coste-camino < alcanzados[s].coste-camino) entonces
        alcanzados[s] ← sucesor
        INSERTAR(sucesor, frontera)
  devolver fallo
```

Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**

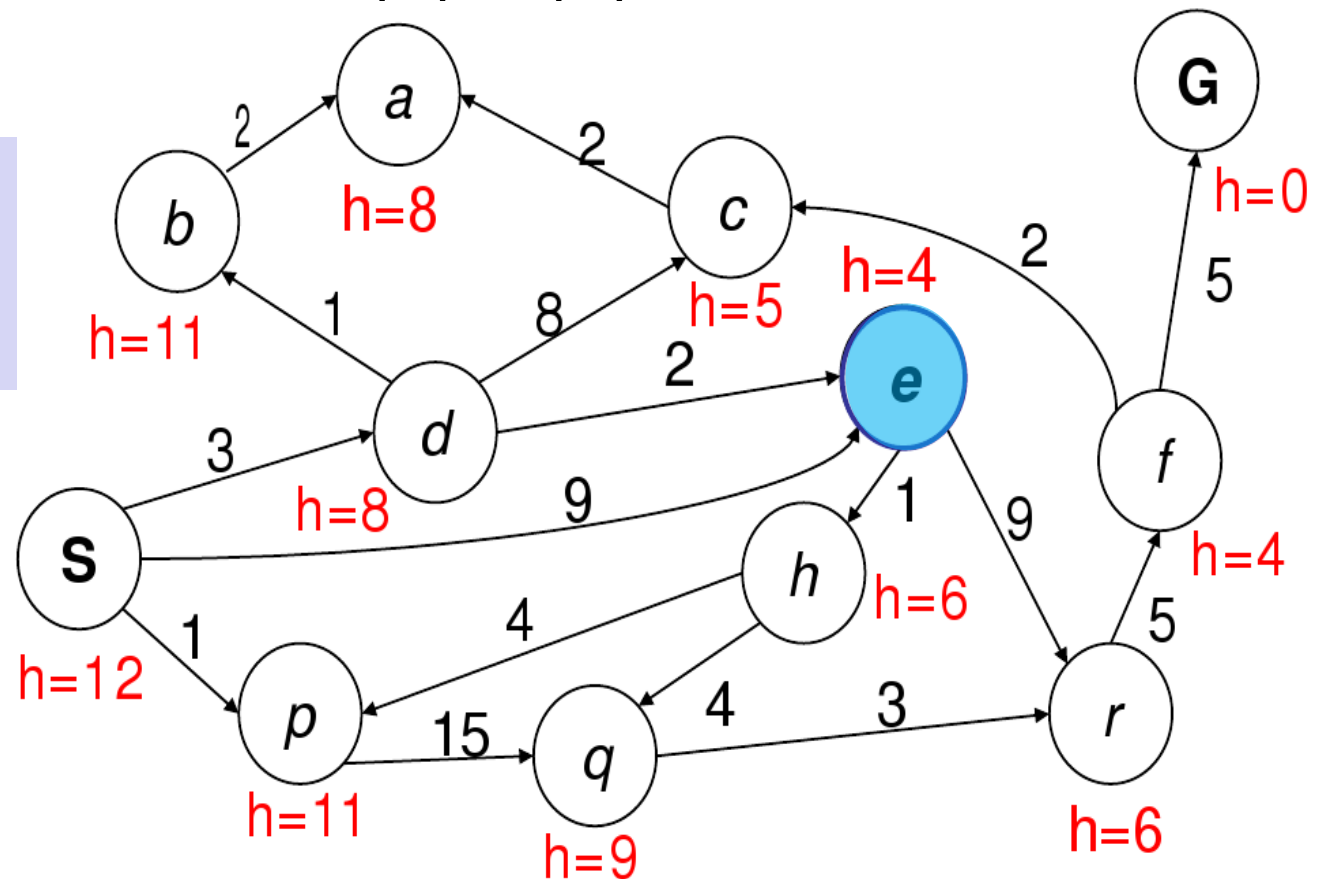


Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**



$al=\{e\}$

$f=\{e\}$

It1: $n=e \neq G$

$al=\{e, h, r\}$

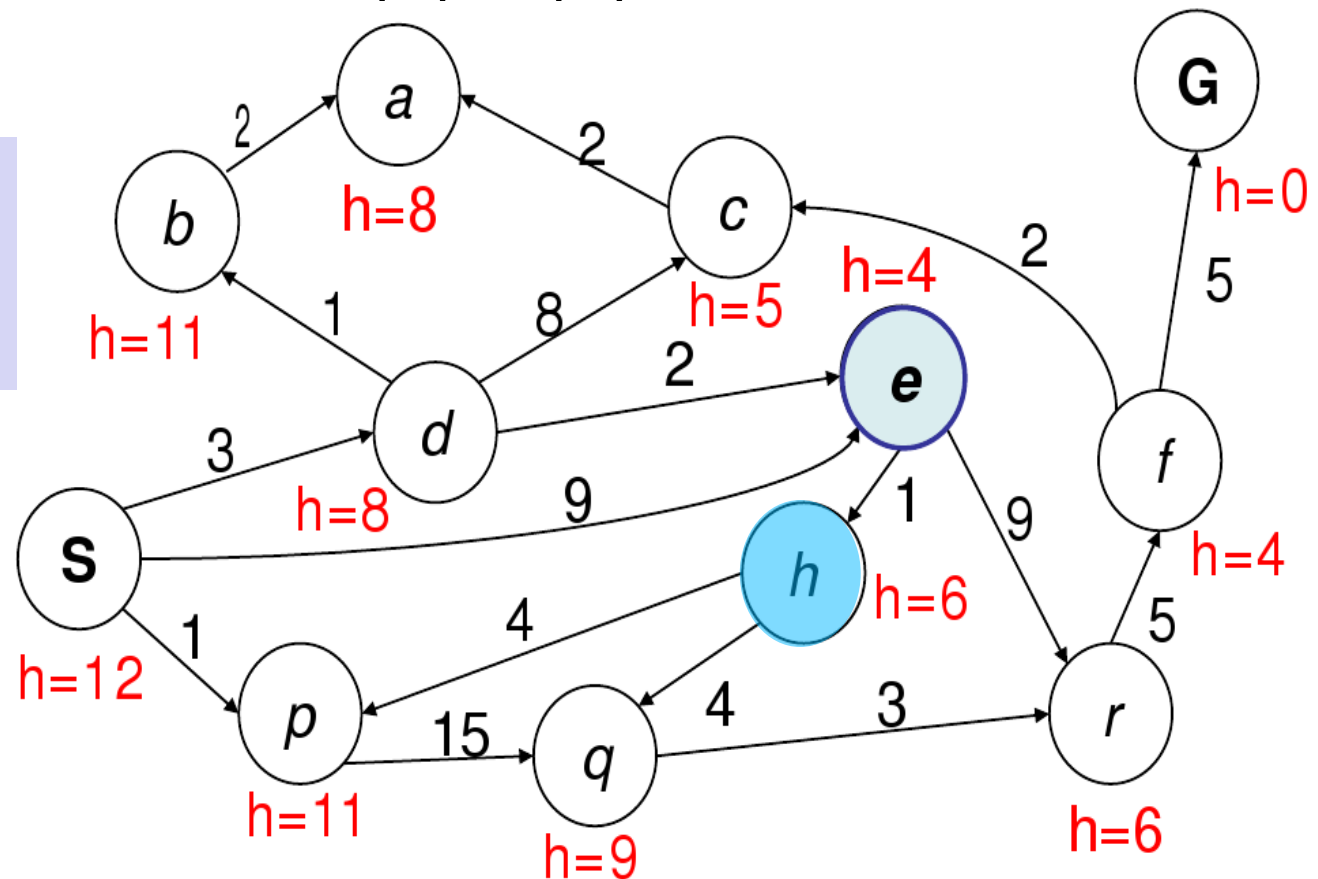
$f=\{h(6), r(6)\}$

Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**



$al=\{e,h,r\}$

$f=\{h(6),r(6)\}$

It2: $n=h \neq G$

$al=\{e,h,r,p,q\}$

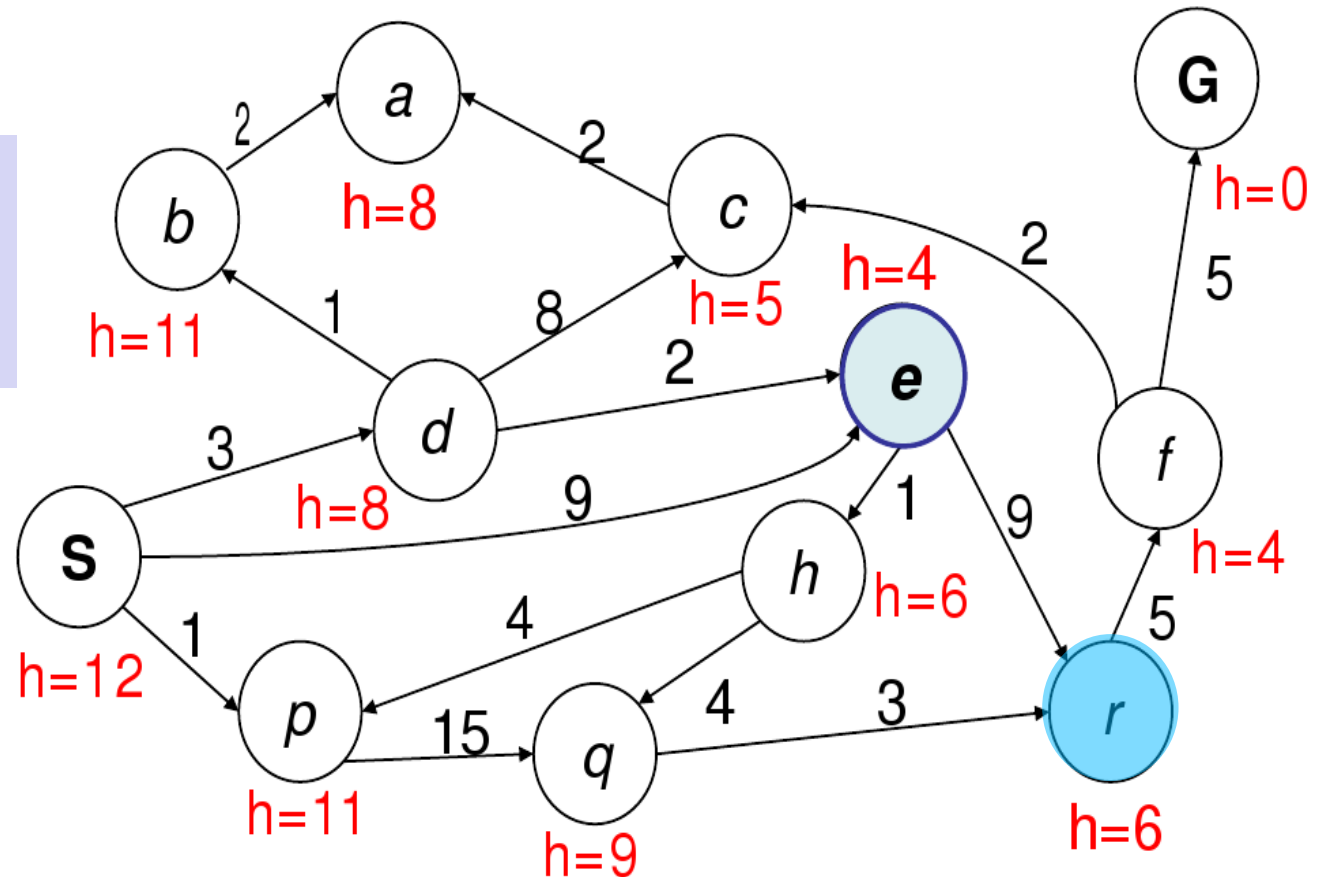
$f=\{r(6),q(9),p(11)\}$

Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**



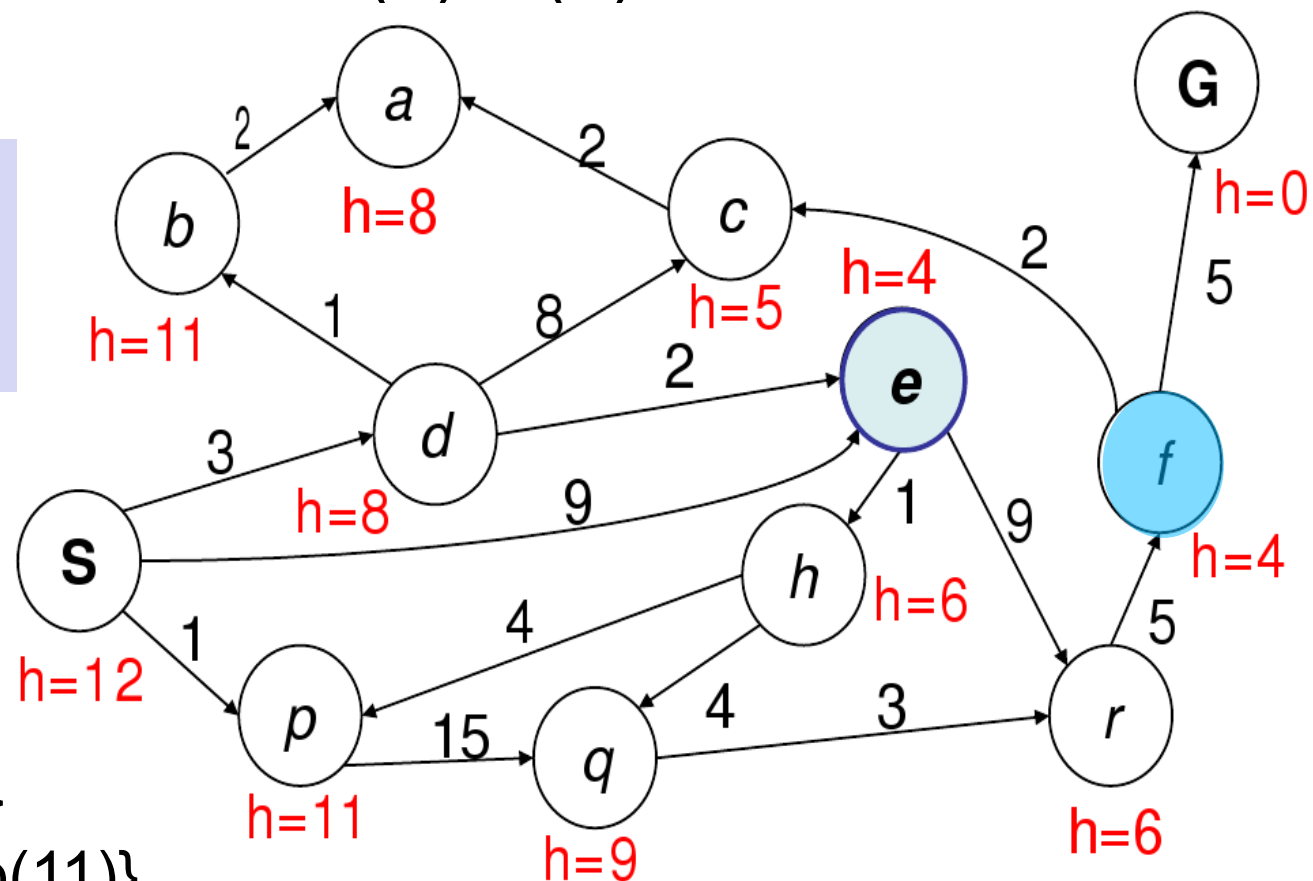
$al=\{e,h,r,p,q\}$
 $f=\{r(6),q(9),p(11)\}$
 It3: $n=r \neq G$
 $al=\{e,h,r,p,q,f\}$
 $f=\{f(4),q(9),p(11)\}$

Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**



$al=\{e,h,r,p,q,f\}$
 $f=\{f(4),q(9),p(11)\}$

It4: $n=f \neq G$

$al=\{e,h,r,p,q,f,c,G\}$
 $f=\{G(0),c(5),q(9),p(11)\}$

Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

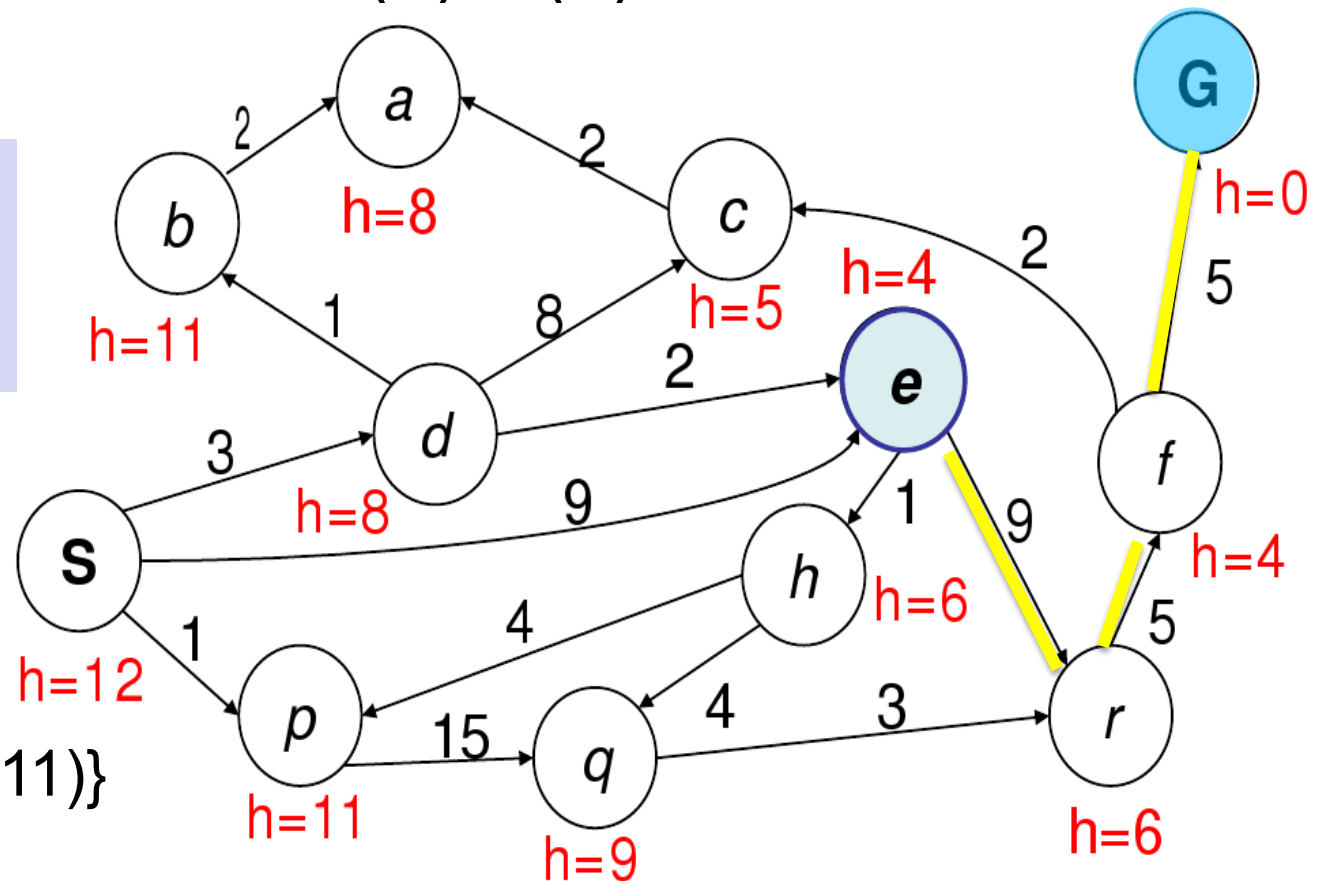
- Nodo inicial: **e**
- Nodo objetivo: **G**

$al=\{e,h,r,p,q,f,c,G\}$

$f=\{G(0),c(5),q(9),p(11)\}$

It5: $n=G=G$

solución(G)=**<irR,irF,irG>**

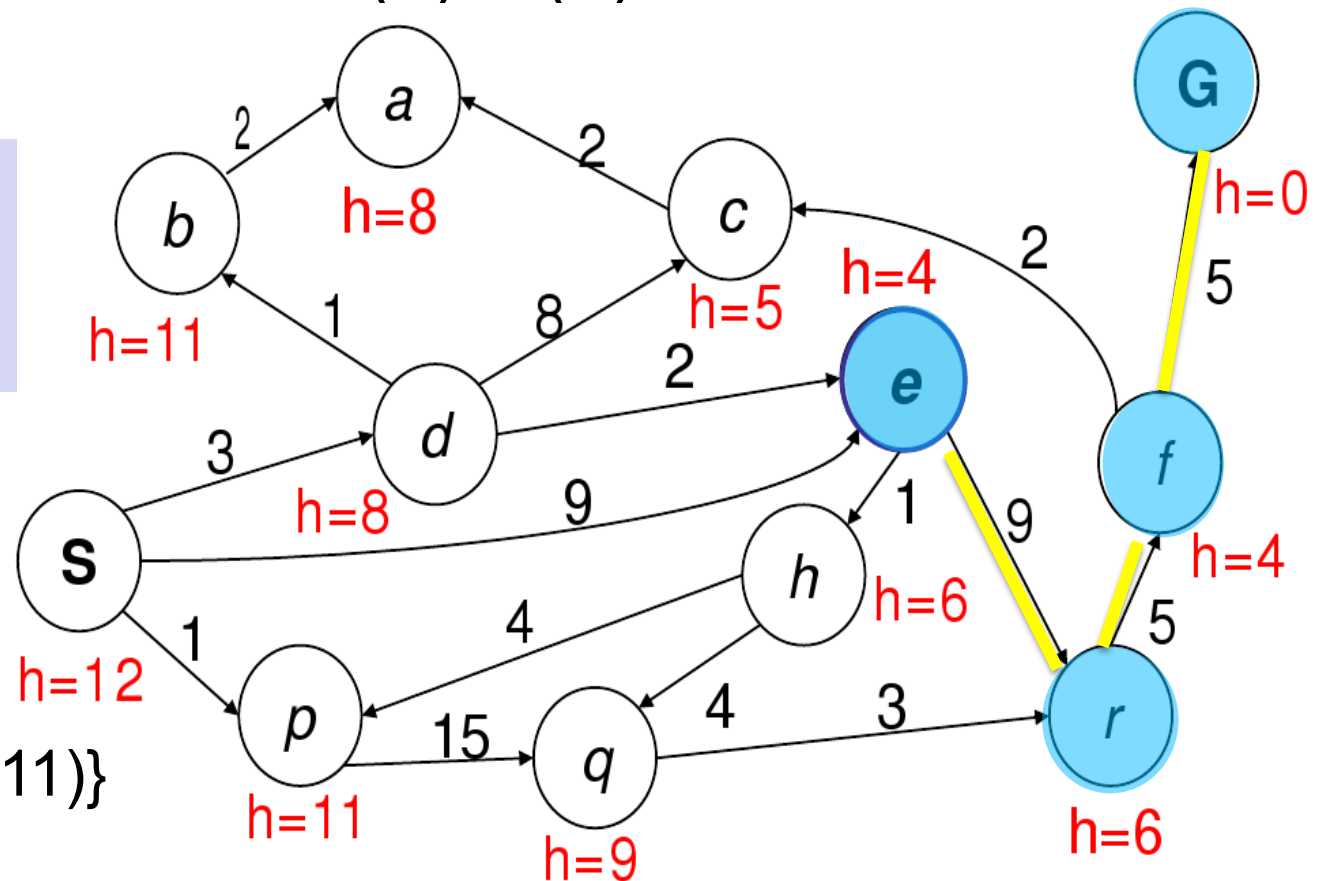


Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**



$al=\{e,h,r,p,q,f,c,G\}$

$f=\{G(0),c(5),q(9),p(11)\}$

It5: $n=G=G$

$\text{solución}(G)=\langle \text{irR}, \text{irF}, \text{irG} \rangle$

$\text{coste}=9+5+5=19$

Búsqueda greedy primero mejor

Inteligencia Artificial

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza $h(n)$)
- Función de evaluación $f(n)=h(n)$

Solución?

- Nodo inicial: **e**
- Nodo objetivo: **G**

$al=\{e,h,r,p,q,f,c,G\}$

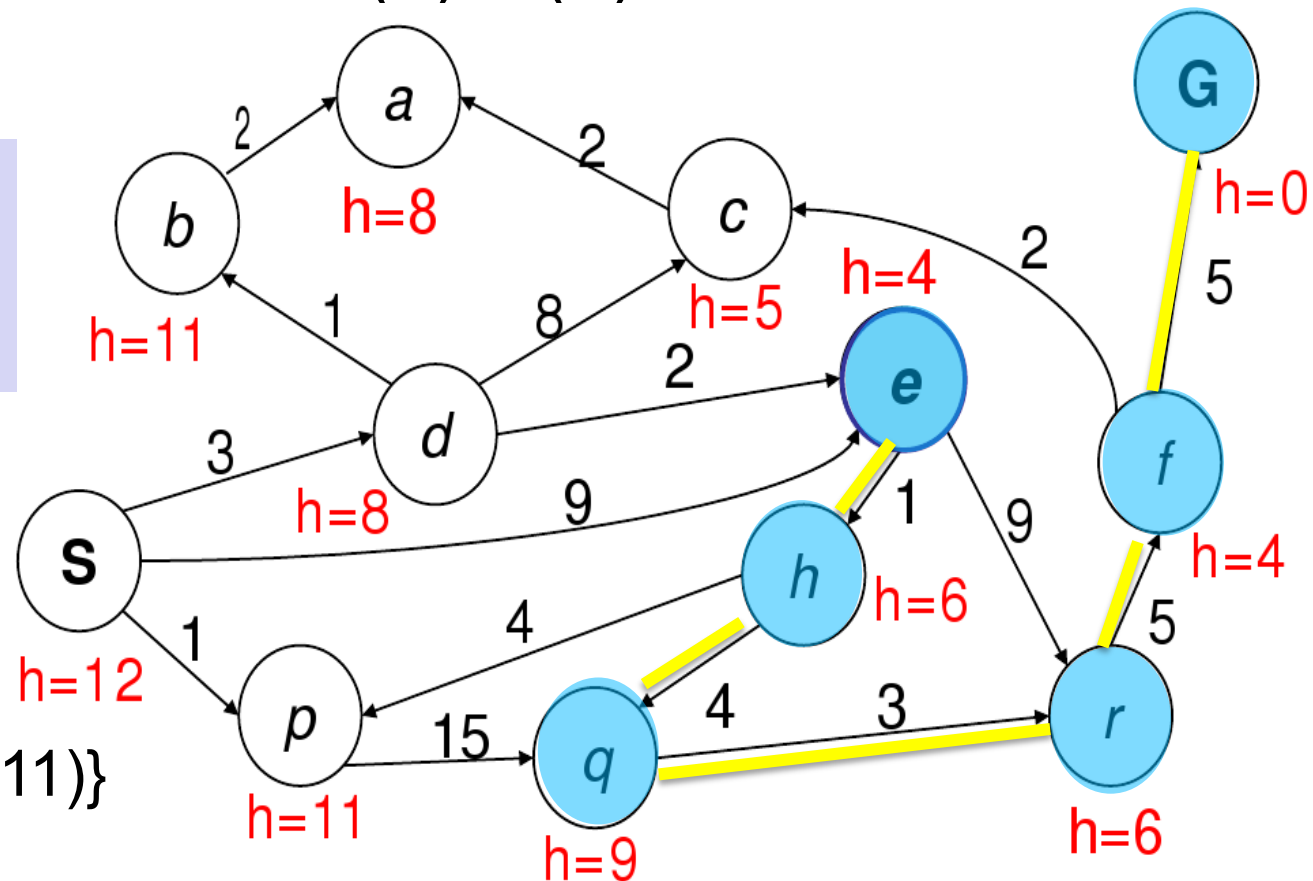
$f=\{G(0),c(5),q(9),p(11)\}$

It5: $n=G=G$

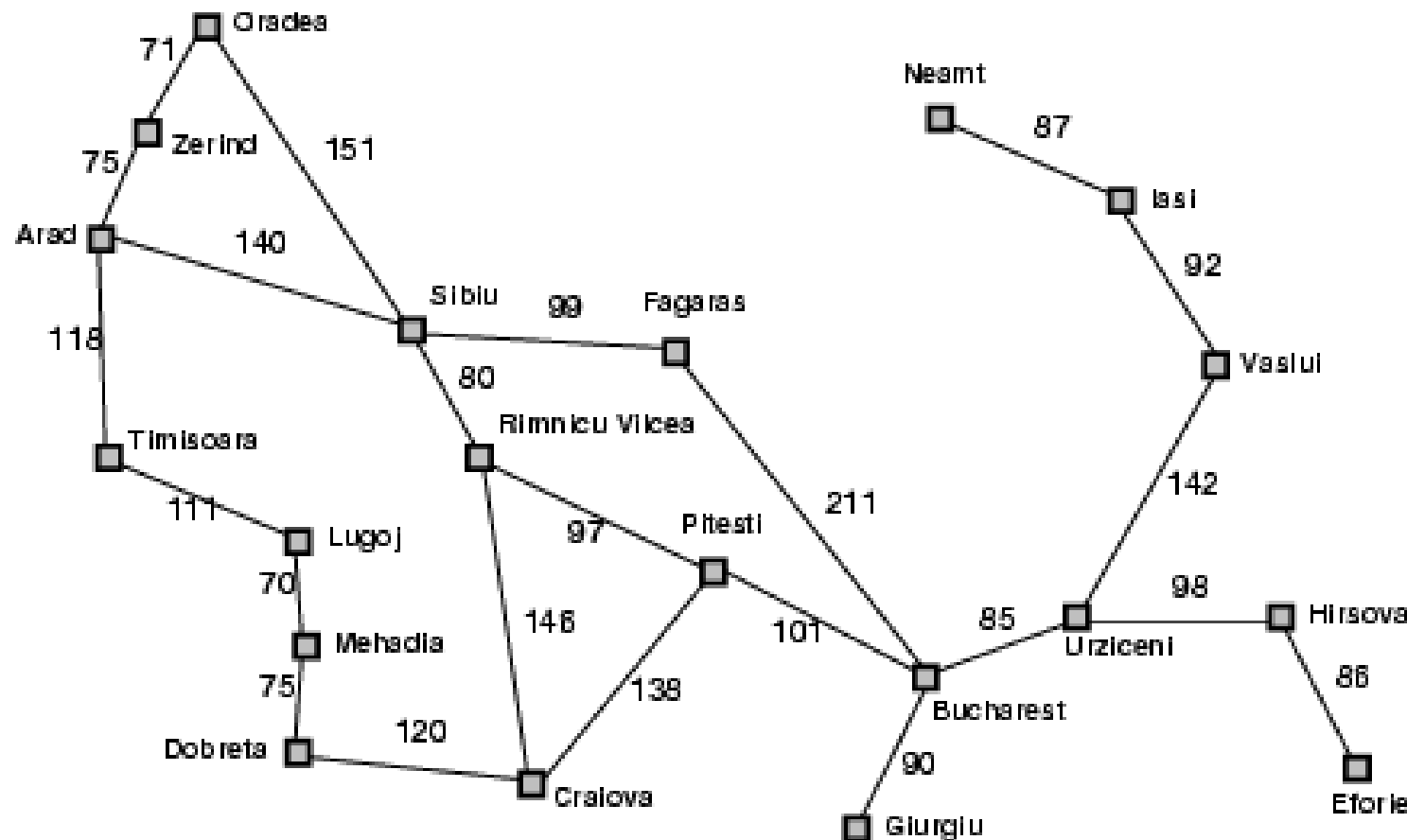
$\text{solución}(G)=\langle irR,irF,irG \rangle$

$\text{coste}=9+5+5=19$

$\text{coste}(\langle irH,irQ,irR,irF,irG \rangle)=1+4+3+5+5=18$



Romania (distancias en km)



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

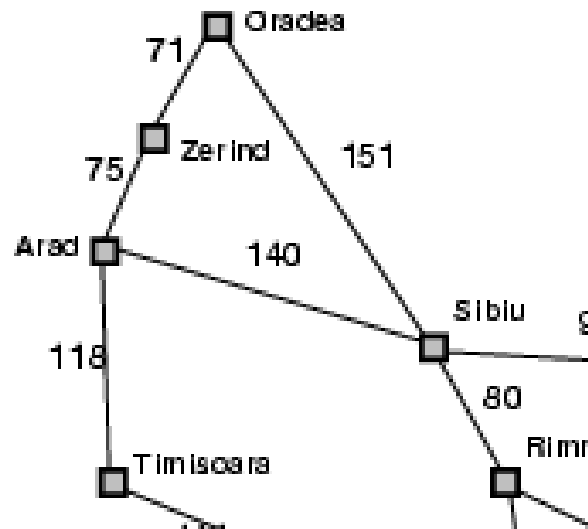
Ej. búsqueda greedy primero mejor

Inteligencia Artificial



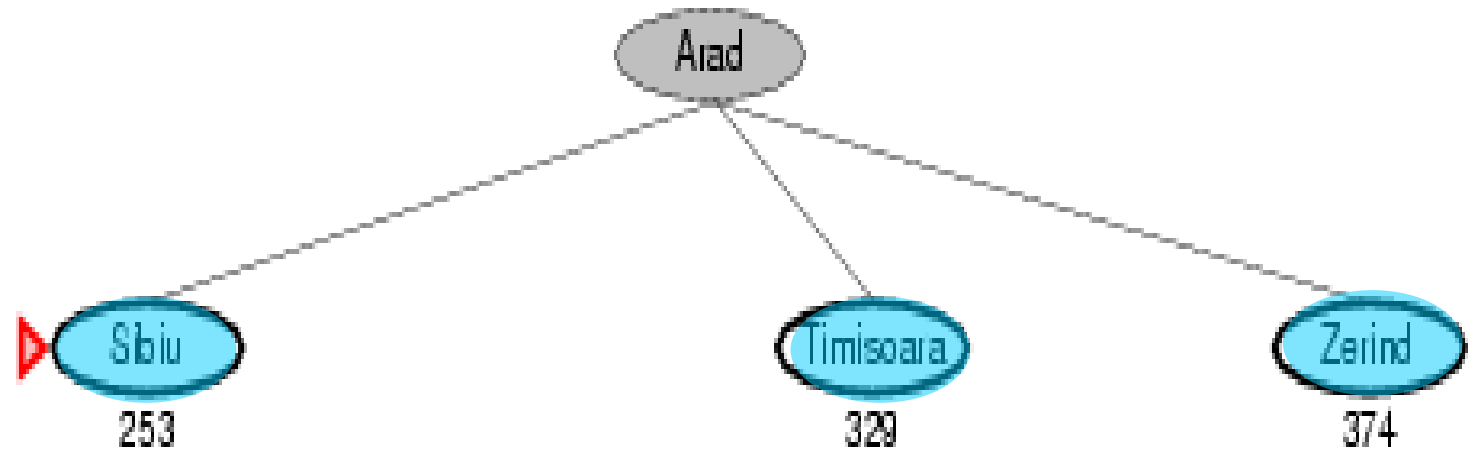
frontera

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

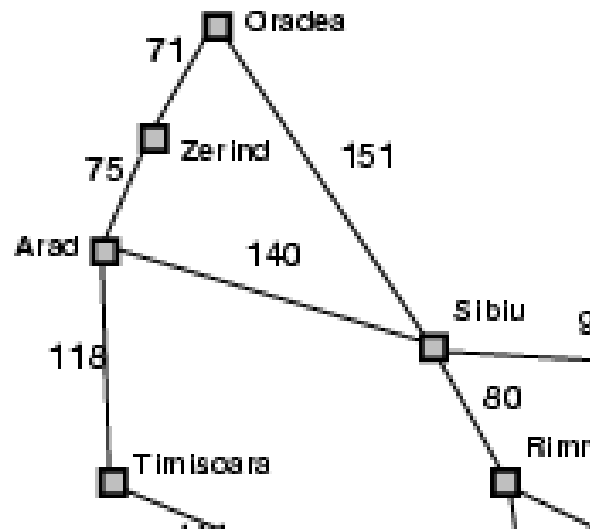


Ej. búsqueda greedy primero mejor

Inteligencia Artificial



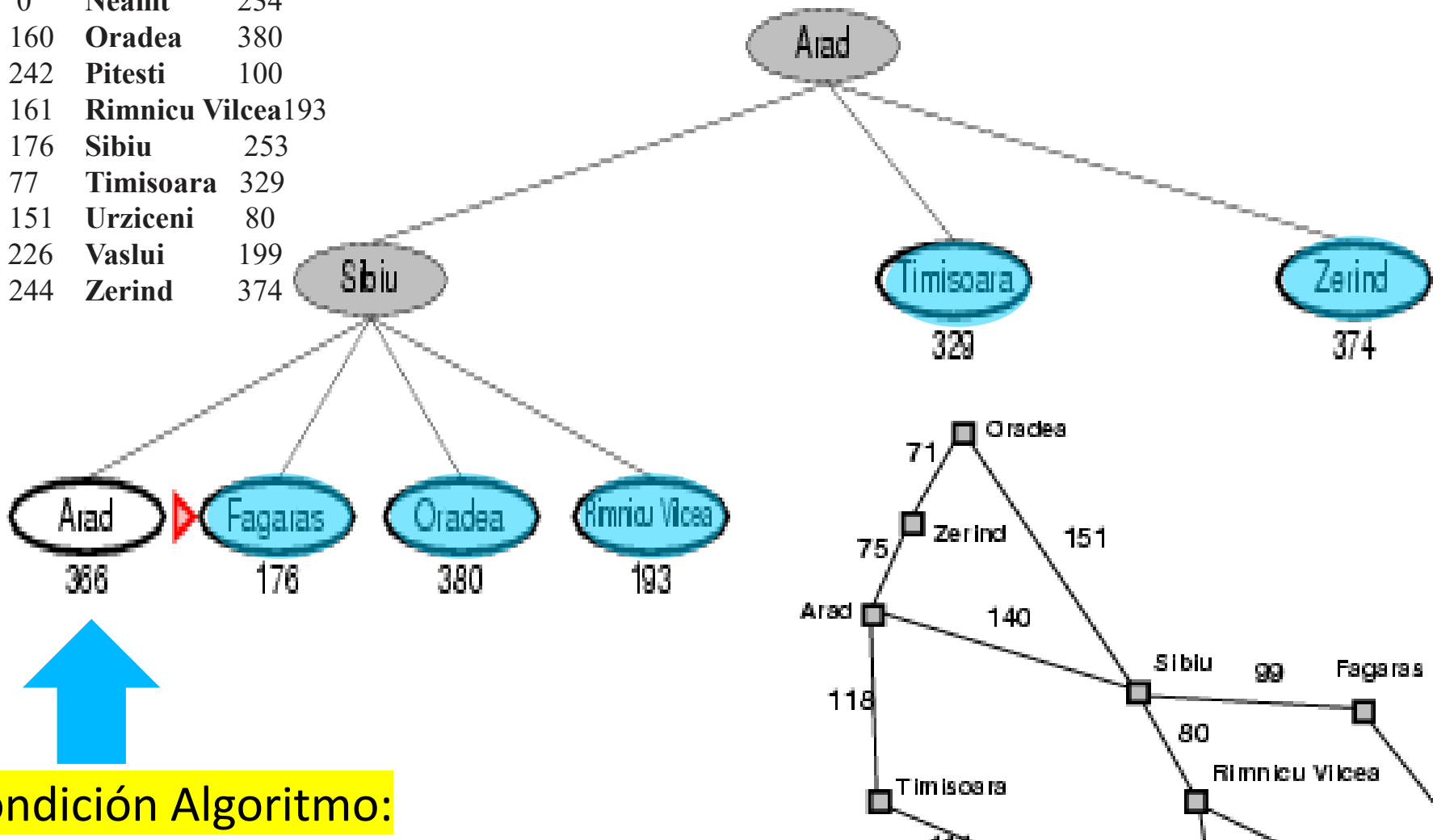
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Ej. búsqueda greedy primero mejor

Inteligencia Artificial

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



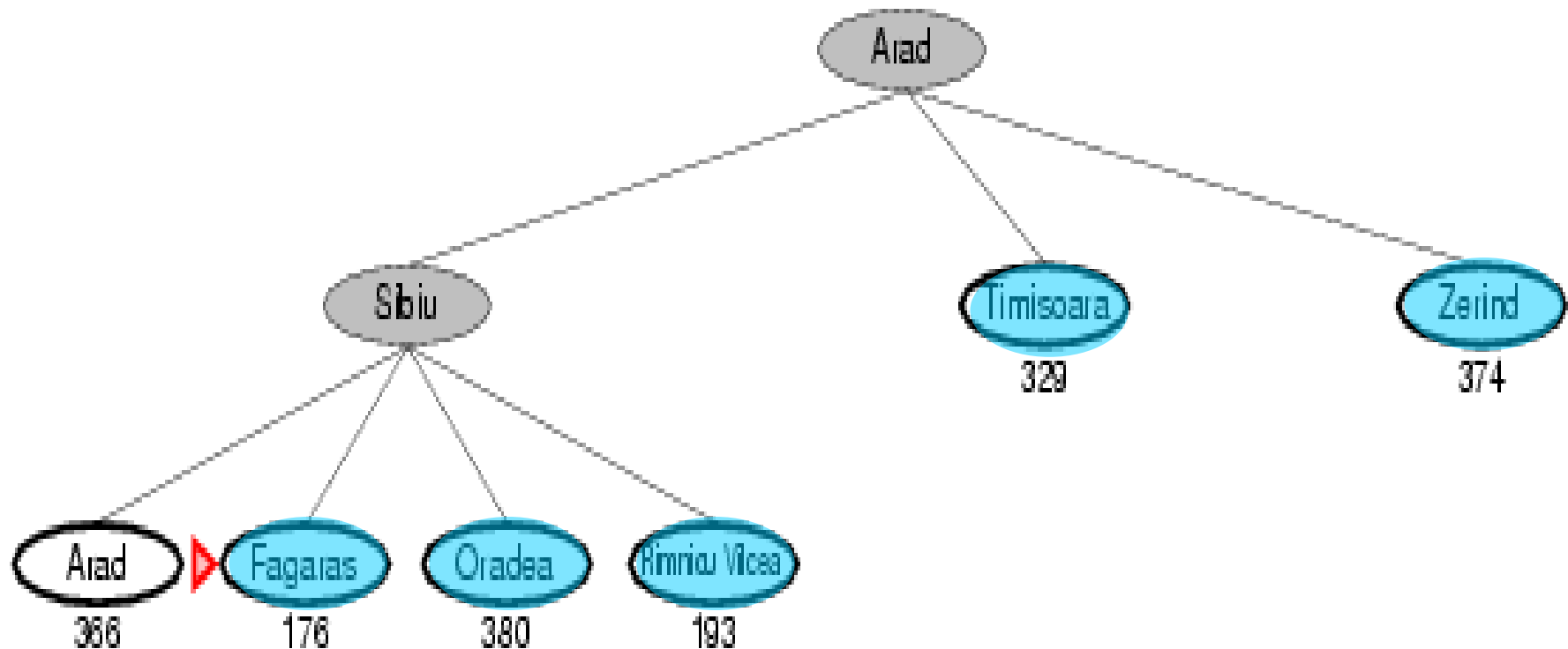
Condición Algoritmo:

si Arad' no está en alcanzados o $\text{coste}(\text{Arad}') < \text{coste}(\text{Arad})$

? < ?

Ej. búsqueda greedy primero mejor

Inteligencia Artificial



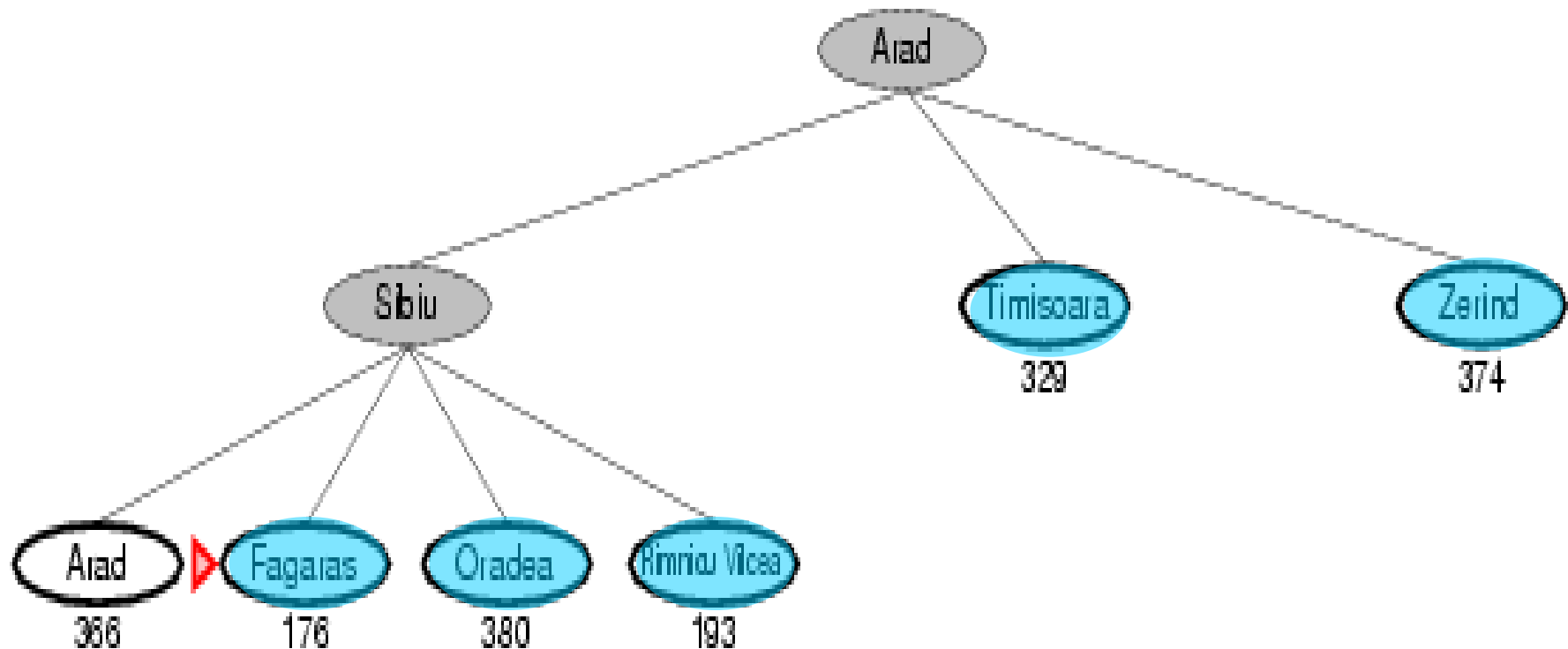
Condición Algoritmo:

si $Arad'$ no está en alcanzados o $coste(Arad') < coste(Arad)$

$280 < 0$

Ej. búsqueda greedy primero mejor

Inteligencia Artificial

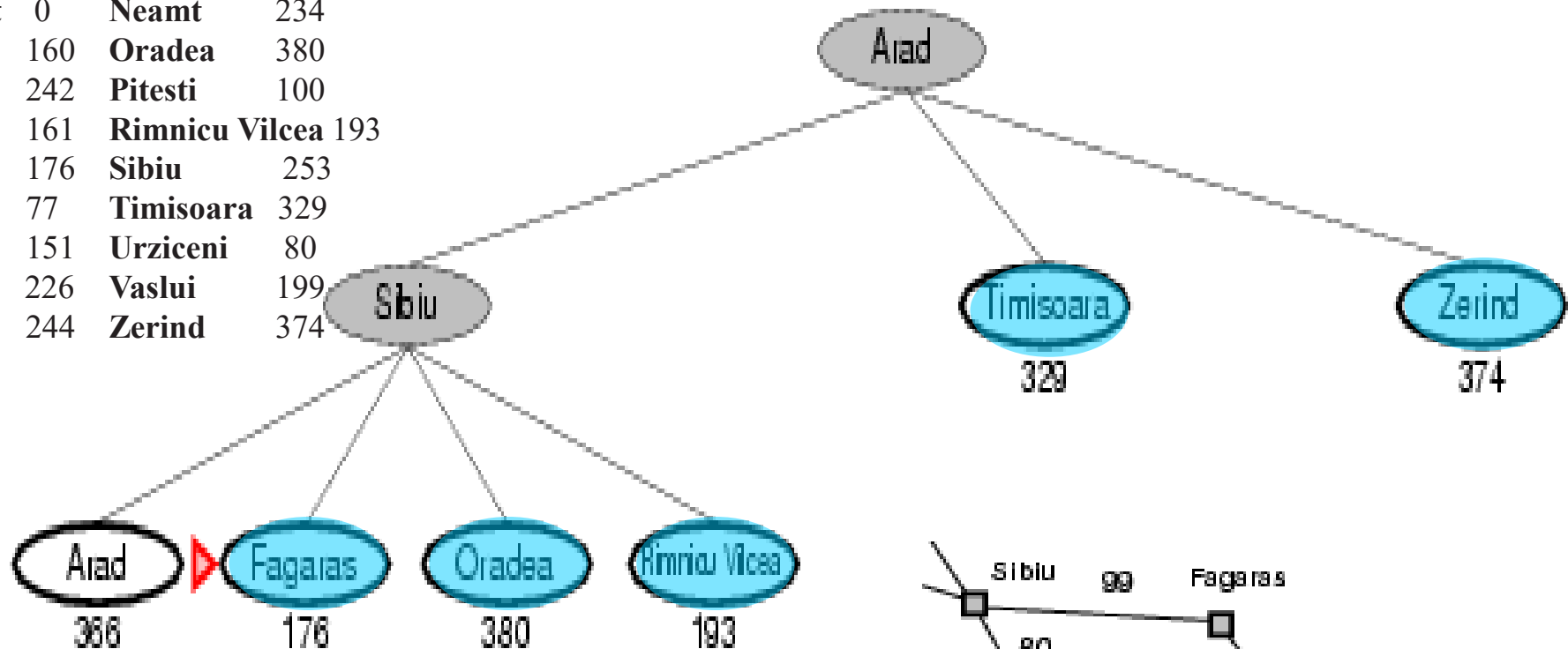


Arad' no entra en la frontera

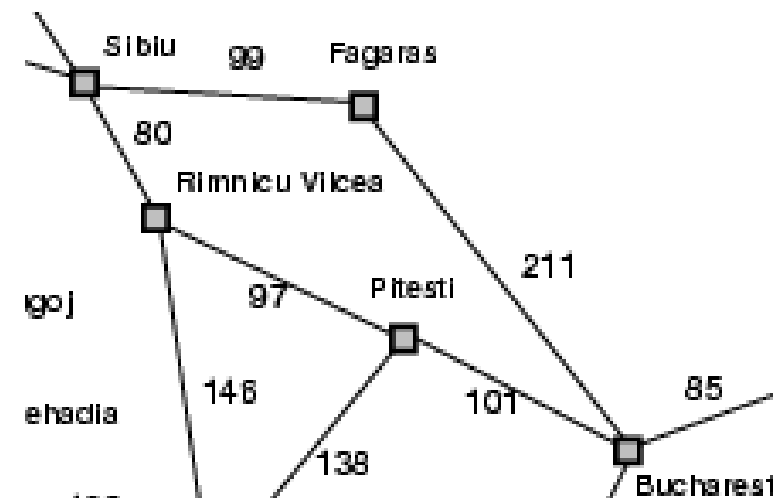
Ej. búsqueda greedy primero mejor

Inteligencia Artificial

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



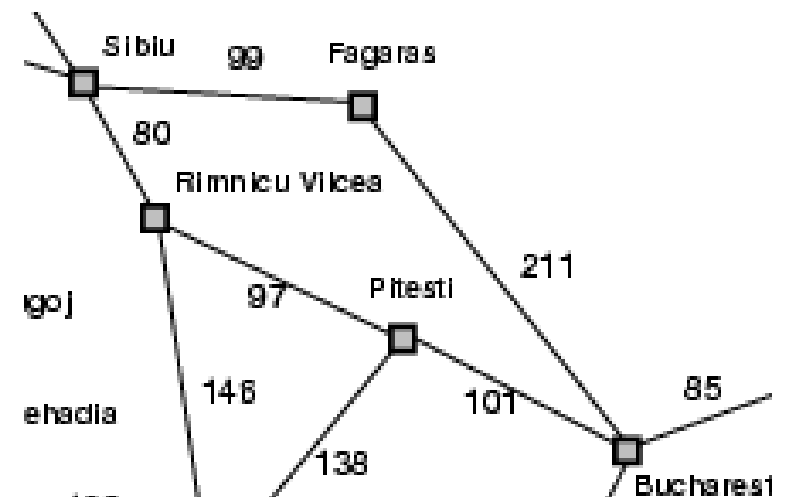
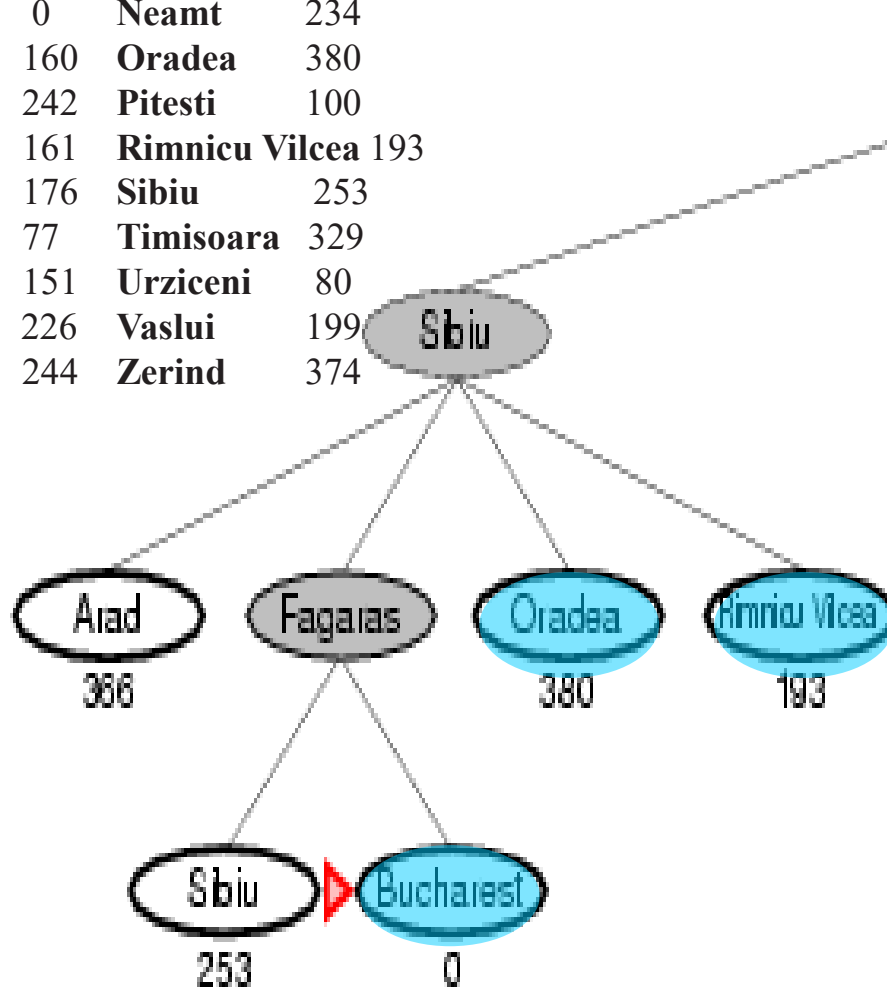
Arad' no entra en la frontera



Ej. búsqueda greedy primero mejor

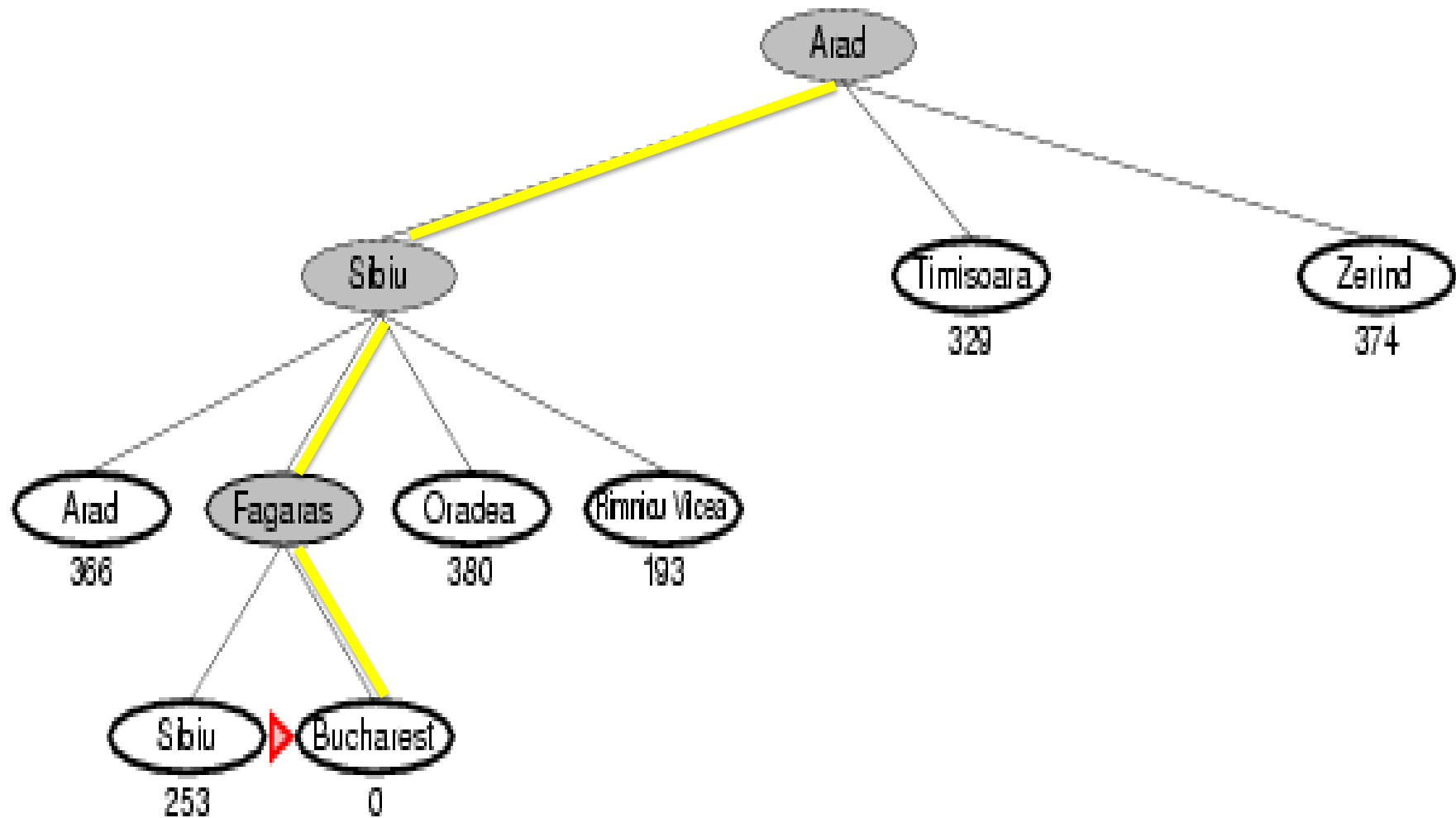
Inteligencia Artificial

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



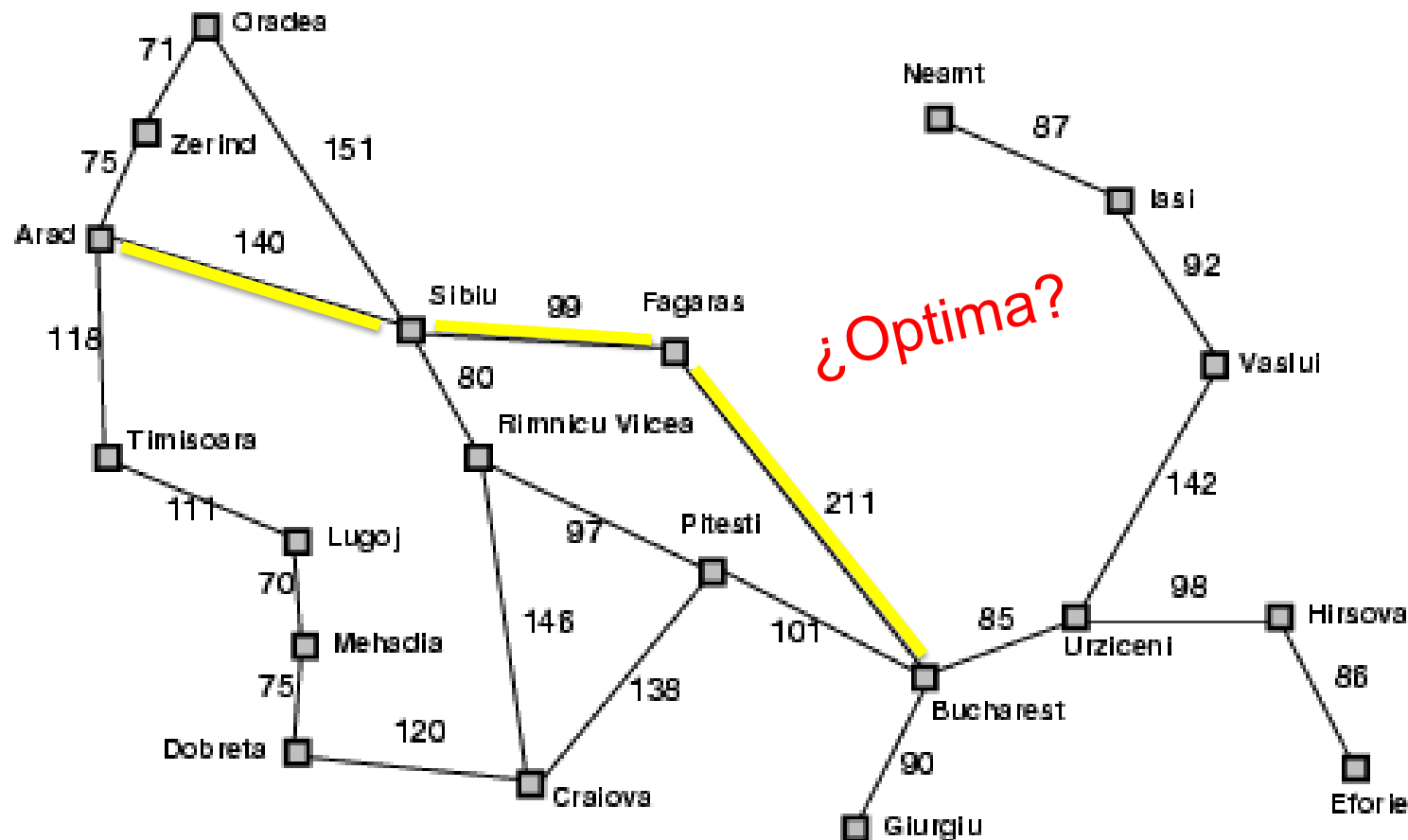
Sibiu' no entra en la frontera: $g(\text{Sibiu}') = 338 > g(\text{Sibiu}) = 140$

Ej. búsqueda greedy primero mejor



Ej. búsqueda greedy primero mejor

Inteligencia Artificial



Sol.=<irSibiu, irFagaras, irBuchar.>

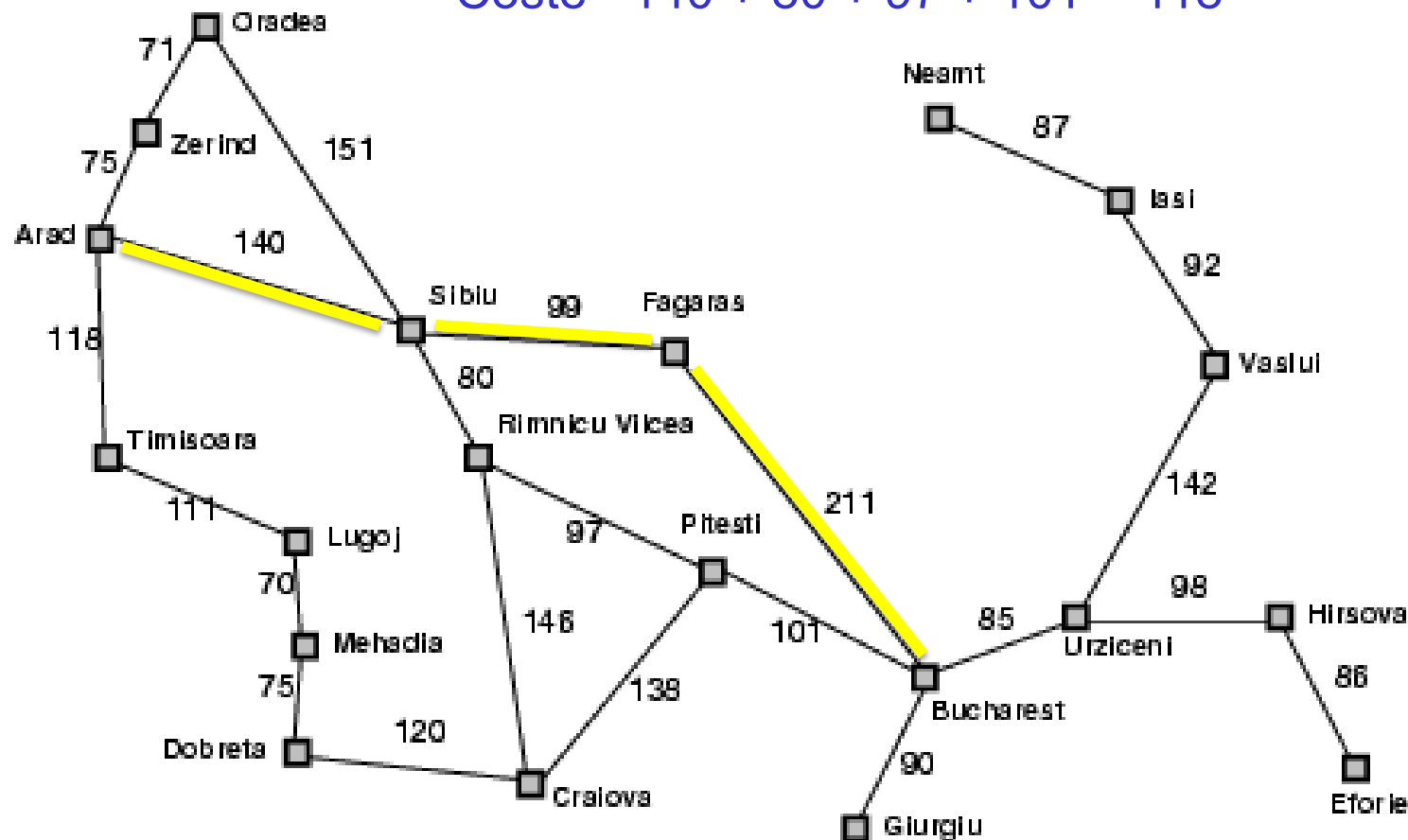
Coste= 140 + 99 + 211 = 450

Ej. búsqueda greedy primero mejor

Inteligencia Artificial

Sol.=<irSibiu, irRimnicu Vikea, irPitesti, irBuchar.>

Coste= 140 + 80 + 97 + 101 = 418



Sol.=<irSibiu, irFagaras, irBuchar.>

Coste= 140 + 99 + 211 = 450

Búsqueda A*

- Idea: evitar expandir caminos que ya son caros
- Función de evaluación $f(n) = g(n) + h(n)$
- $g(n)$ = coste de alcanzar n desde el estado inicial
- $h(n)$ = coste estimado desde n hasta el objetivo
- $f(n)$ = coste estimado de la mejor solución que pase por n

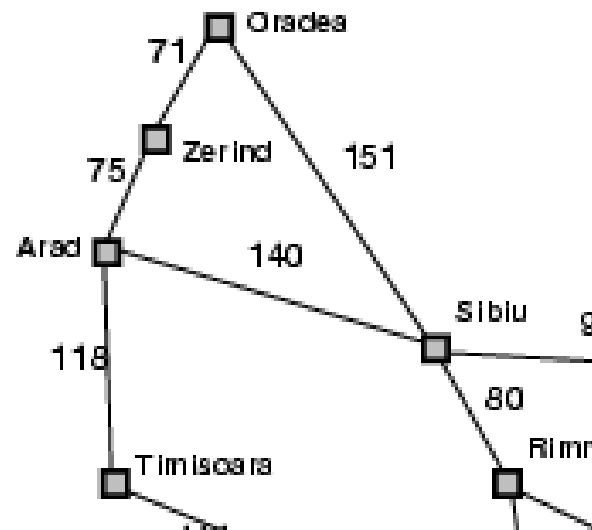
Ejemplo de búsqueda A*

Inteligencia Artificial



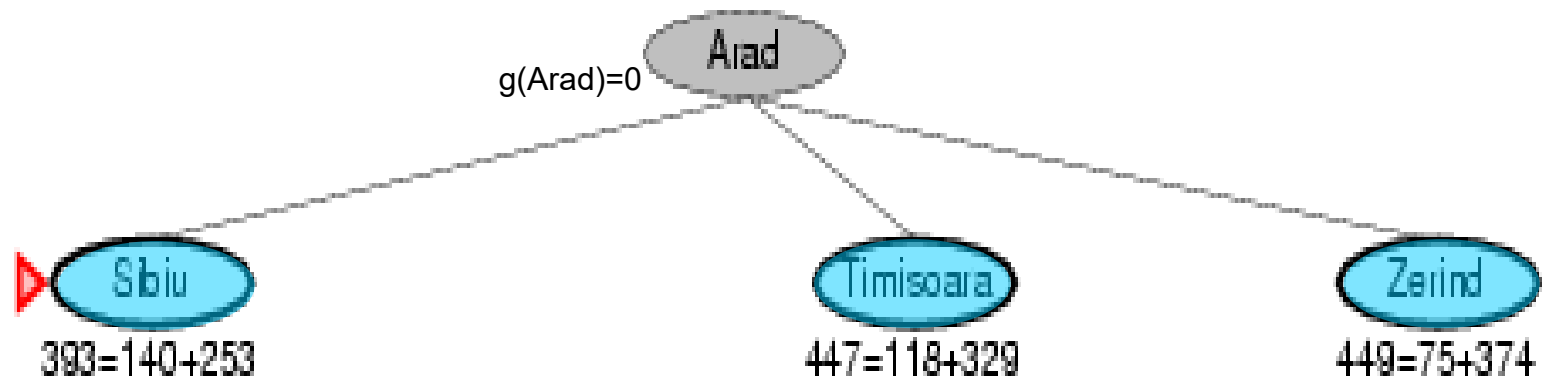
frontera

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

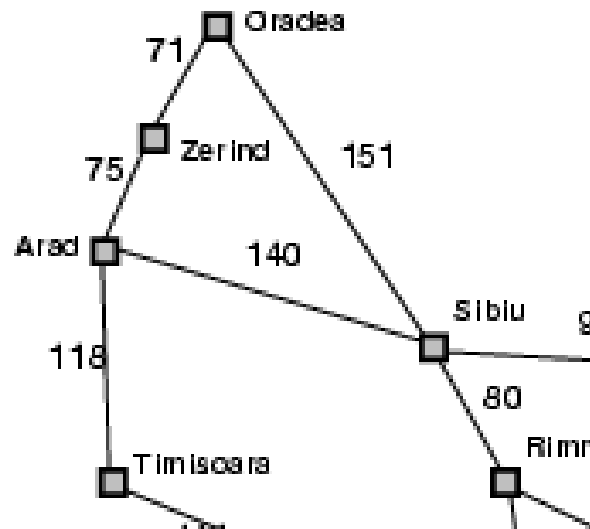


Ejemplo de búsqueda A*

Inteligencia Artificial



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

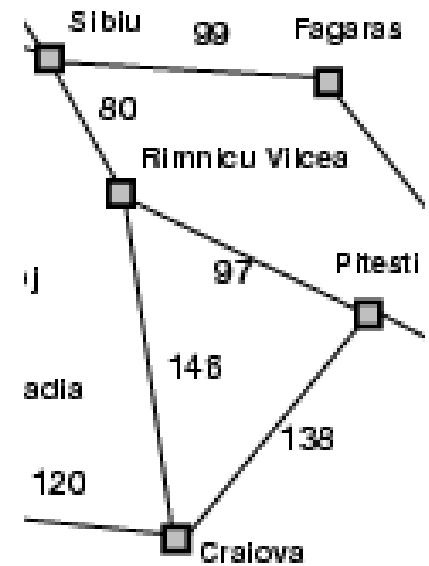
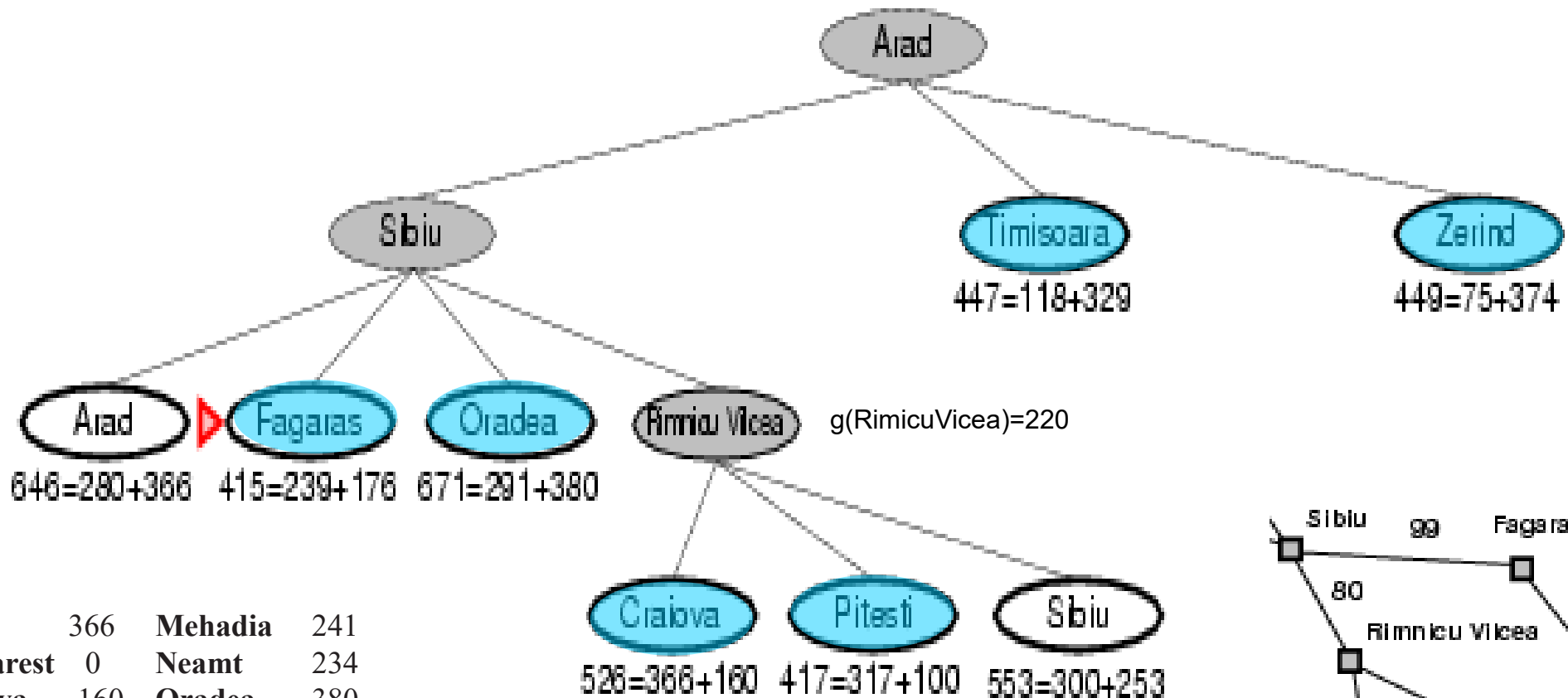


```

graph TD
    Oradea ---|71| Zerind
    Oradea ---|151| Sibiu
    Zerind ---|75| Arad
    Arad ---|140| Sibiu
    Arad ---|118| Timisoara
    Sibiu ---|80| Rimnicu Vilcea
    Sibiu ---|99| Fagaras
    Timisoara ---|119| Lugoj
    Rimnicu Vilcea ---|93| Pitesti
    Fagaras ---|87| Bucharest
    
```

Ejemplo de búsqueda A*

Inteligencia Artificial

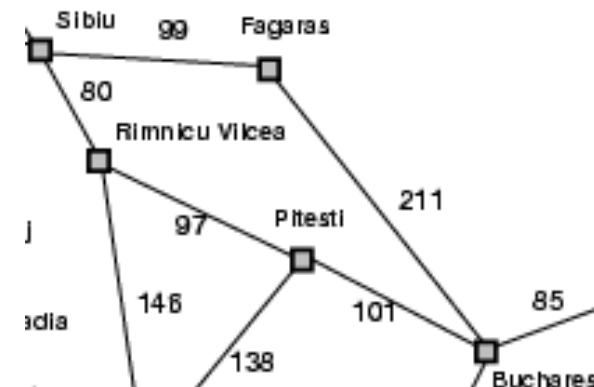
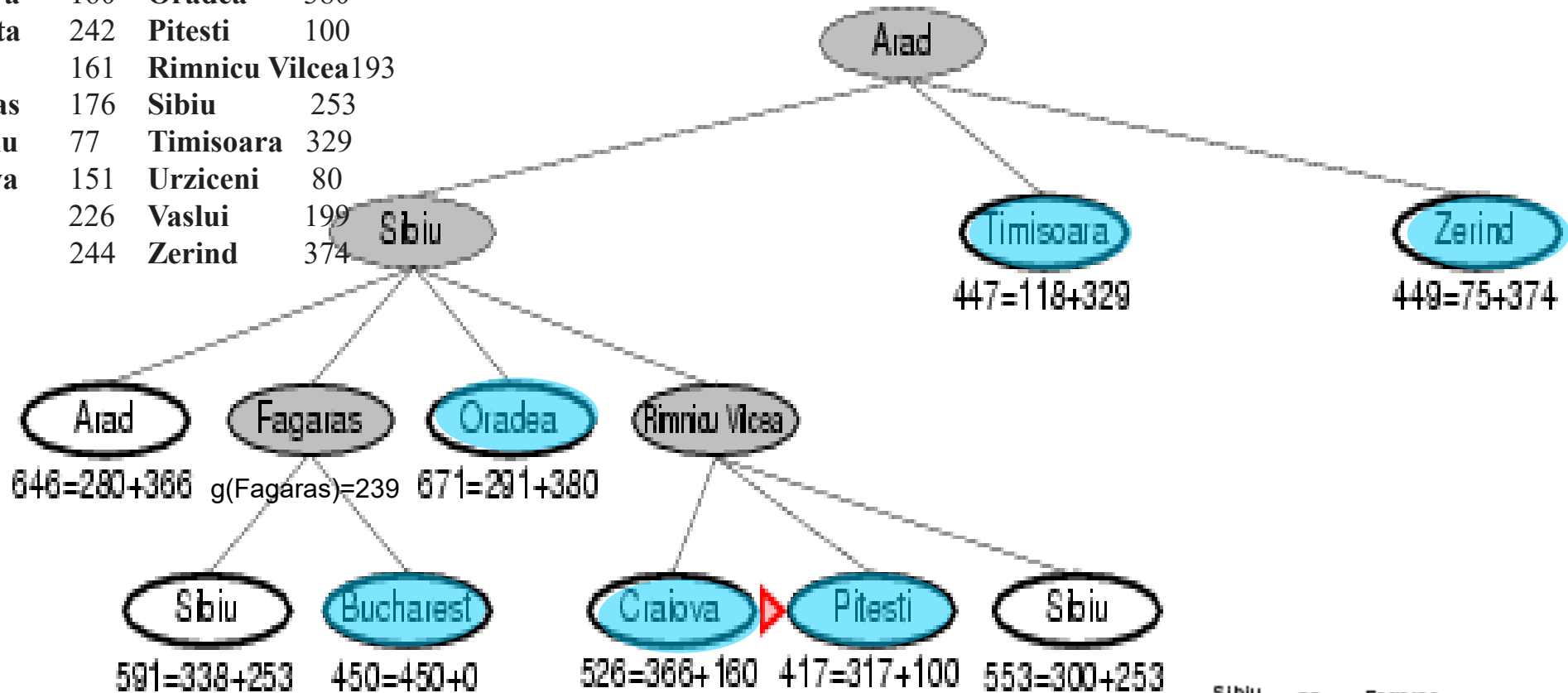


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Ejemplo de búsqueda A*

Inteligencia Artificial

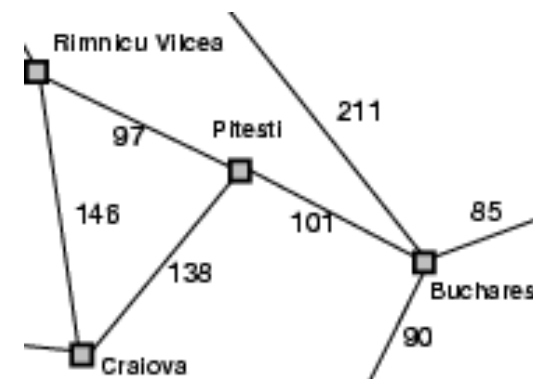
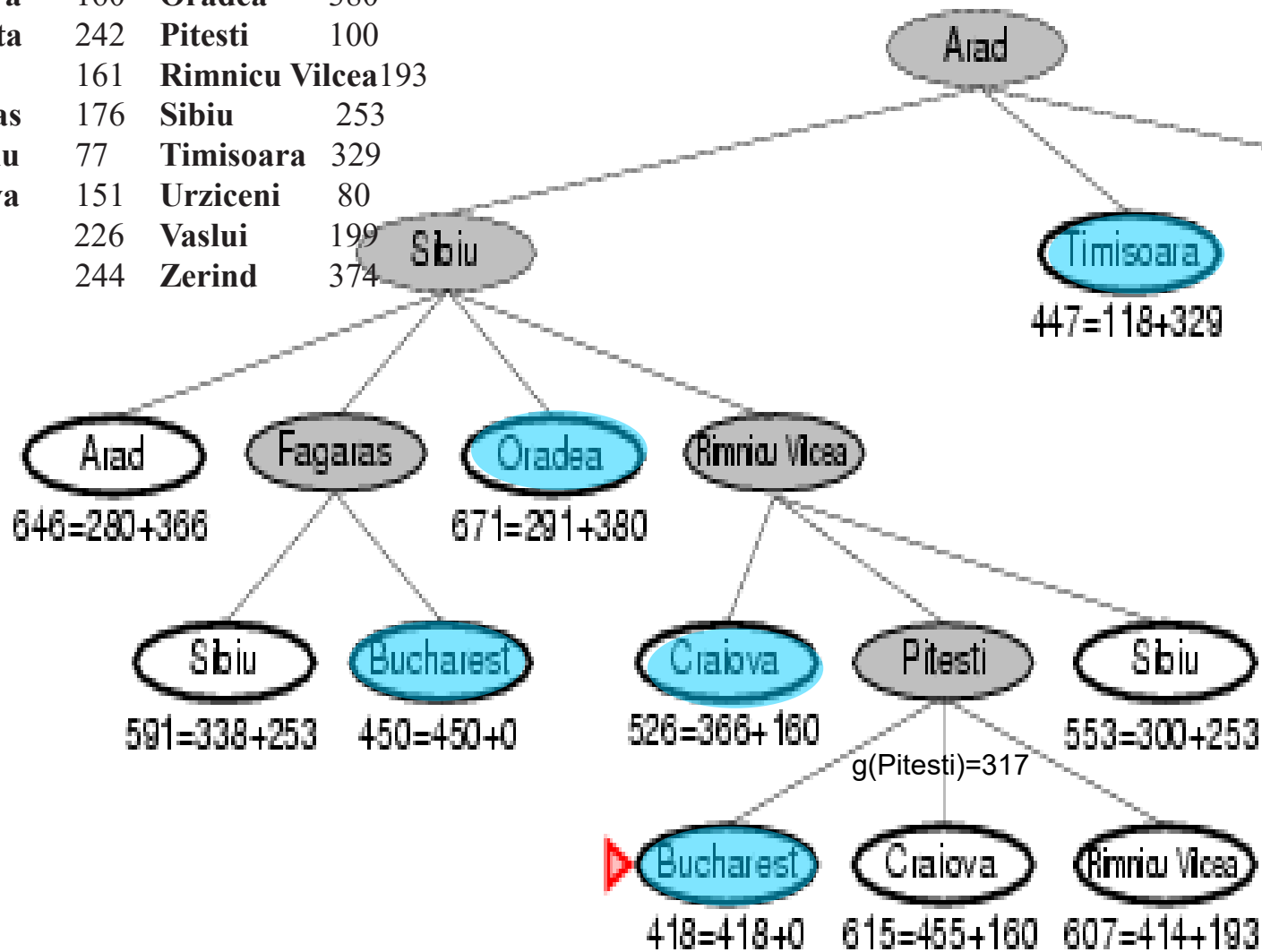
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Ejemplo de búsqueda A*

Inteligencia Artificial

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



Ejemplo de búsqueda A*

función BEST-FIRST-SEARCH(*problema*, *f*) **devuelve** una solución o fallo

nodo \leftarrow CREAM-NODO(*problema.estado-inicial*), *alcanzados* \leftarrow \emptyset , *frontera* \leftarrow \emptyset

INSERTAR((*problema.estado-inicial*, nodo), *alcanzados*)

INSERTAR(nodo, *frontera*)

bucle while not VACIA? (*frontera*) **hacer**

nodo \leftarrow SACAR-BORRANDO-PRIMERO(*frontera*)

si ES-OBJETIVO (nodo.estado) **entonces devolver** SOLUCION(nodo)

bucle for each *sucesor* en EXPANDIR(nodo, *problema*) **hacer**

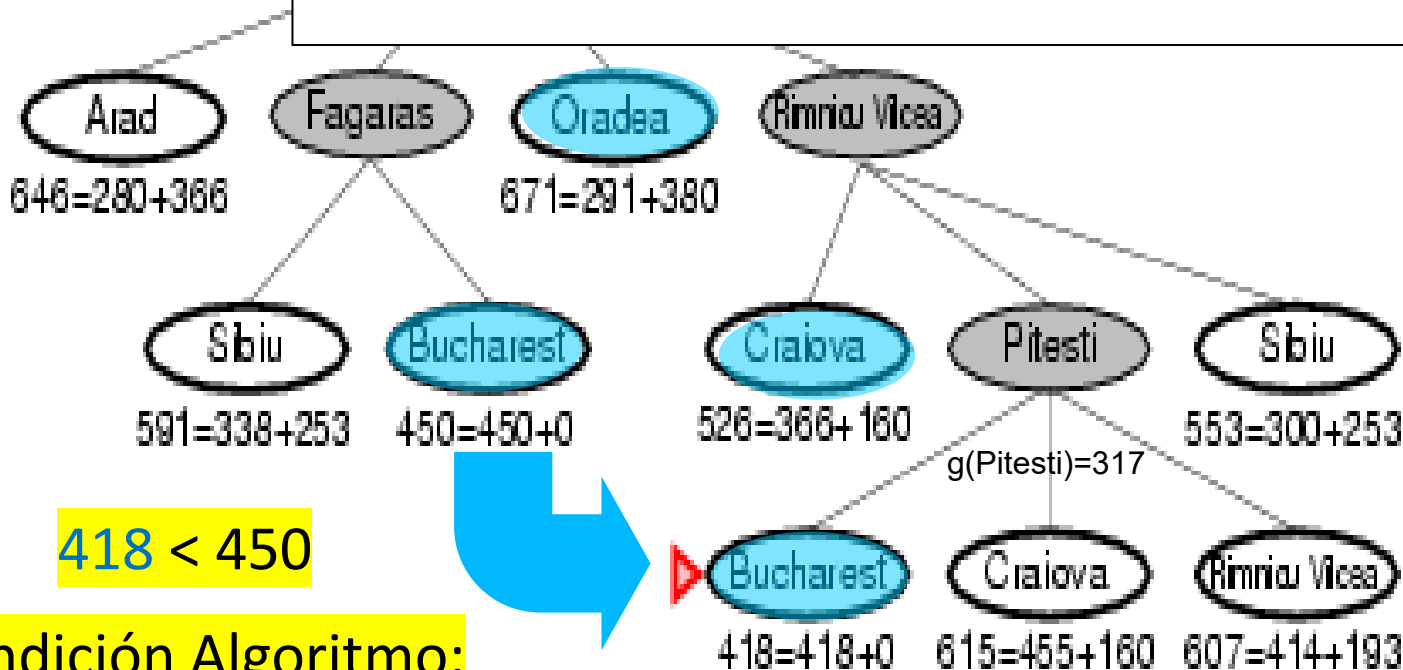
s \leftarrow *sucesor.estado*

si *s* no está en *alcanzados* o (*sucesor.coste-camino* < *alcanzados*[*s*].coste-camino) **entonces**

alcanzados[*s*] \leftarrow *sucesor*

INSERTAR(*sucesor*, *frontera*)

devolver fallo

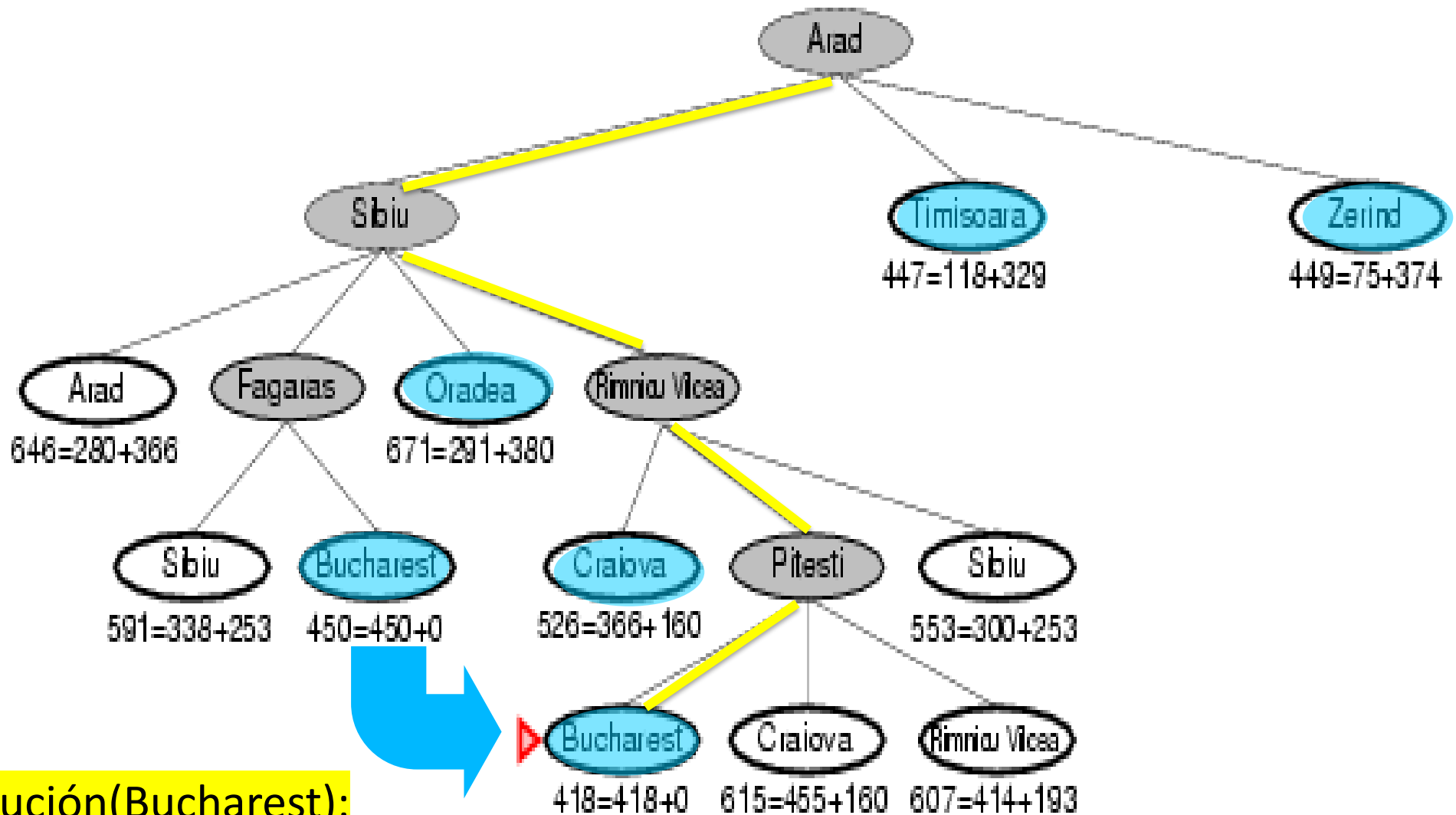


418 < 450

Condición Algoritmo:

si Bucharest' no está en alcanzados o coste(Bucharest') < coste(Bucharest)

Ejemplo de búsqueda A*



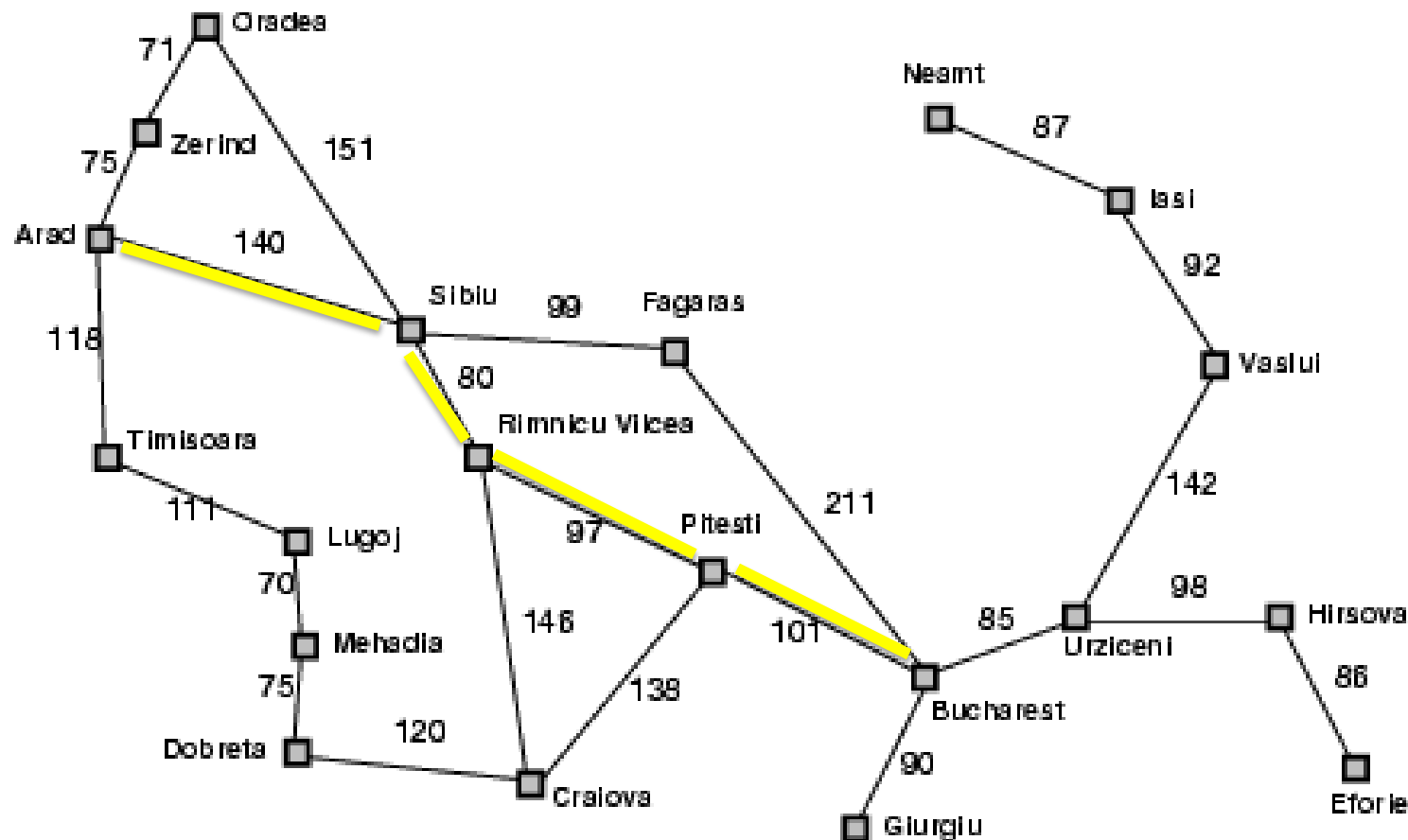
Solución(Bucharest):

<irSibiu, irRimicuVicea, irPitesti, irBucharest>

Coste: 418

Ejemplo de búsqueda A*

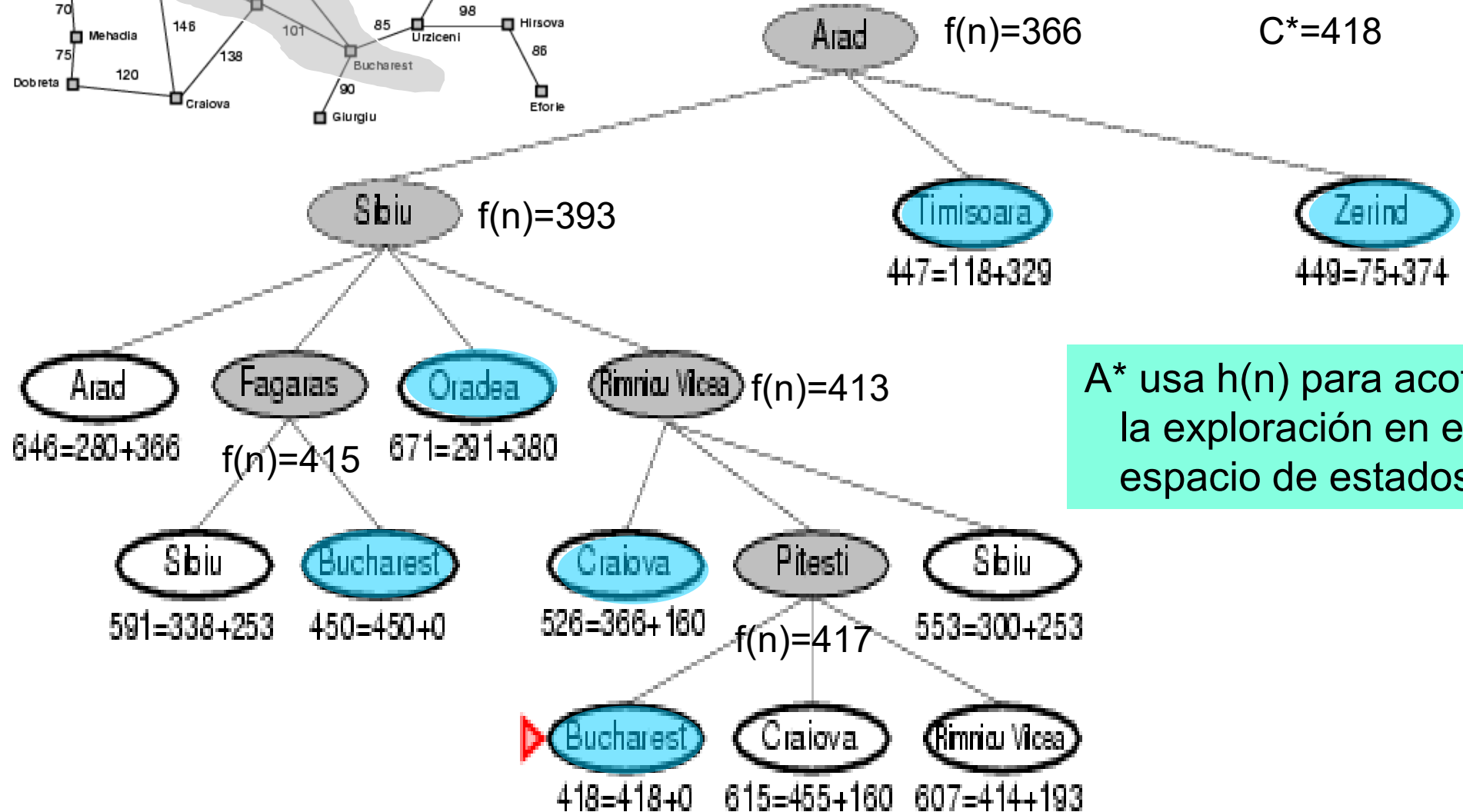
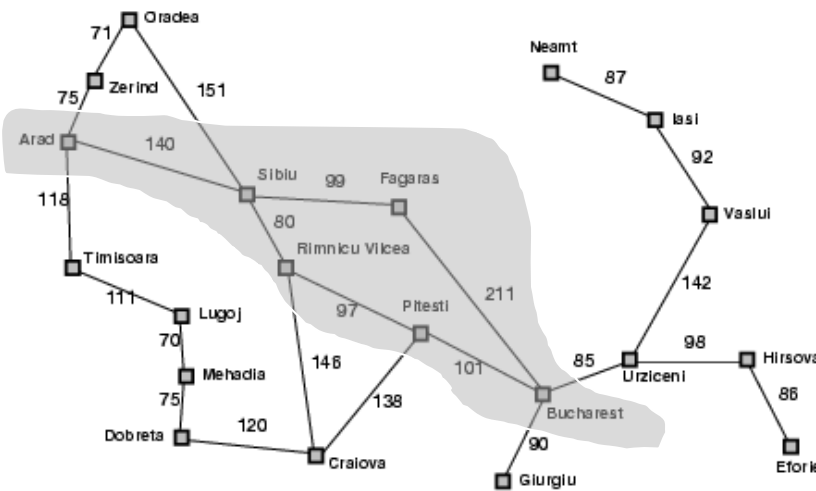
Inteligencia Artificial



Sol.=<irSibiu, irRimnicu Vikea, irPitesti, irBuchar.>

Coste= 140 + 80 + 97 + 101 = 418

la búsqueda A*



A* usa $h(n)$ para acotar la exploración en el espacio de estados

Heurísticas admisibles

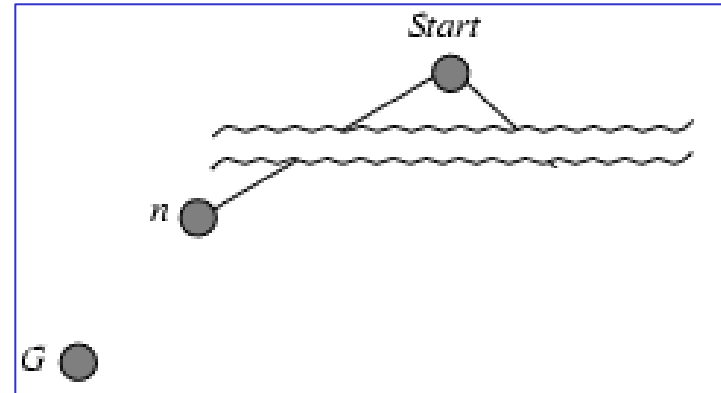
- Una heurística $h(n)$ es **admisible** si para todo nodo n , $h(n) \leq h^*(n)$, donde $h^*(n)$ es el coste **real** de alcanzar el estado objetivo desde n .
- Una heurística admisible **nunca sobre-estima** el coste de alcanzar el objetivo, es decir, es **optimista**
- Ejemplo: $h_{SLD}(n)$ nunca sobre-estima la distancia por carretera

Teorema: Si $h(n)$ es admisible, la búsqueda A^* usando ***búsqueda en árbol*** es óptima

Optimalidad de A^* (prueba)

Prueba por contradicción:

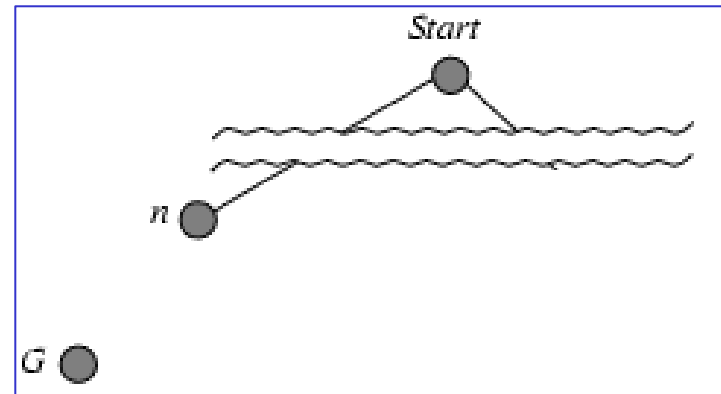
- Suponer que el camino óptimo tiene coste C^* pero que el algoritmo retorna un camino con coste $C > C^*$.
- Sea n un nodo no expandido en la frontera que pertenece a un camino óptimo.
- $g^*(n)$: coste del camino óptimo desde el inicio hasta n
- $h^*(n)$: coste del camino óptimo desde n hasta el objetivo más cercano



Optimalidad de A^* (prueba)

Prueba por contradicción:

- Suponer que el camino óptimo tiene coste C^* pero que el algoritmo retorna un camino con coste $C > C^*$.
- Sea n un nodo no expandido en la frontera que pertenece a un camino óptimo.
- $g^*(n)$: coste del camino óptimo desde el inicio hasta n
- $h^*(n)$: coste del camino óptimo desde n hasta el objetivo más cercano

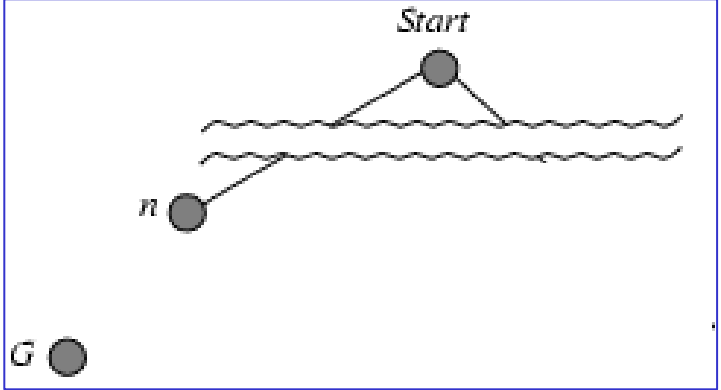


Entonces:

- $f(n) > C^*$ (porque si no, n habría sido expandido)
- $f(n) = g(n) + h(n)$ (por definición)

Optimalidad de A^* (prueba)

Prueba por contradicción:

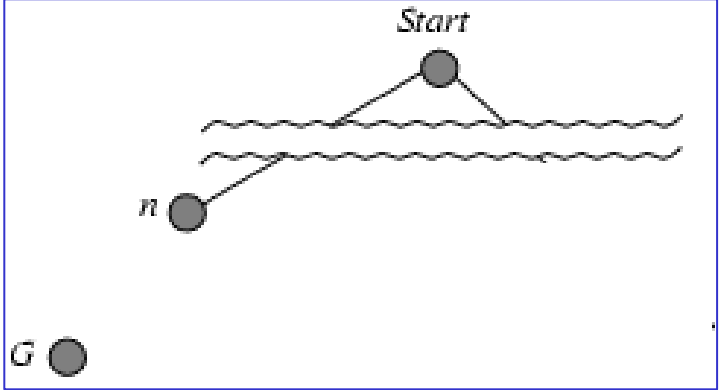
- Suponer que el camino óptimo tiene coste C^* pero que el algoritmo retorna un camino con coste $C > C^*$.
 - Sea n un nodo no expandido en la frontera que pertenece a un camino óptimo.
- 
- $g^*(n)$: coste del camino óptimo desde el inicio hasta n
 - $h^*(n)$: coste del camino óptimo desde n hasta el objetivo más cercano

Entonces:

- $f(n) > C^*$ (porque si no, n habría sido expandido)
- $f(n) = g(n) + h(n)$ (por definición)
- $f(n) = g^*(n) + h(n)$ (porque n está en el camino óptimo)
- $f(n) \leq g^*(n) + h^*(n)$ (por admisibilidad, $h(n) \leq h^*(n)$)

Optimalidad de A^* (prueba)

Prueba por contradicción:

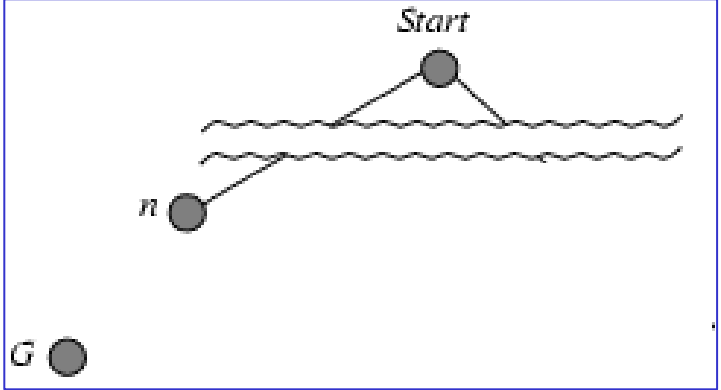
- Suponer que el camino óptimo tiene coste C^* pero que el algoritmo retorna un camino con coste $C > C^*$.
 - Sea n un nodo no expandido en la frontera que pertenece a un camino óptimo.
- 
- $g^*(n)$: coste del camino óptimo desde el inicio hasta n
 - $h^*(n)$: coste del camino óptimo desde n hasta el objetivo más cercano

Entonces:

- $f(n) > C^*$ (porque si no, n habría sido expandido)
- $f(n) = g(n) + h(n)$ (por definición)
- $f(n) = g^*(n) + h(n)$ (porque n está en el camino óptimo)
- $f(n) \leq g^*(n) + h^*(n)$ (por admisibilidad, $h(n) \leq h^*(n)$)
- $f(n) \leq C^*$ (por definición, $C^* = g^*(n) + h^*(n)$)

Optimalidad de A^* (prueba)

Prueba por contradicción:

- Suponer que el camino óptimo tiene coste C^* pero que el algoritmo retorna un camino con coste $C > C^*$.
 - Sea n un nodo no expandido en la frontera que pertenece a un camino óptimo.
- 
- $g^*(n)$: coste del camino óptimo desde el inicio hasta n
 - $h^*(n)$: coste del camino óptimo desde n hasta el objetivo más cercano

Entonces:

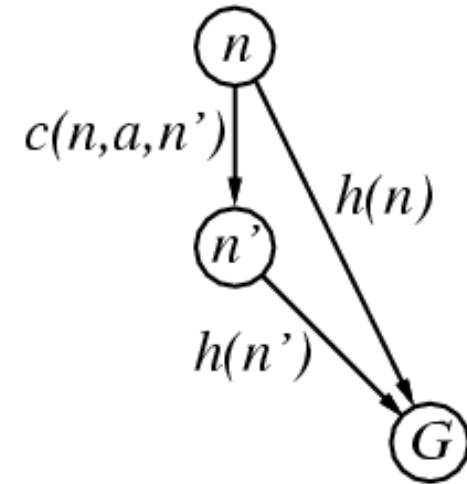
- $f(n) > C^*$ (porque si no, n habría sido expandido)
- $f(n) = g(n) + h(n)$ (por definición)
- $f(n) = g^*(n) + h(n)$ (porque n está en el camino óptimo)
- $f(n) \leq g^*(n) + h^*(n)$ (por admisibilidad, $h(n) \leq h^*(n)$)
- $f(n) \leq C^*$ (por definición, $C^* = g^*(n) + h^*(n)$)



Heurísticas consistentes

- Una heurística es **consistente** si para cada nodo n , todo sucesor n' de n generado por cualquier acción a , cumple que:

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



Heurísticas consistentes

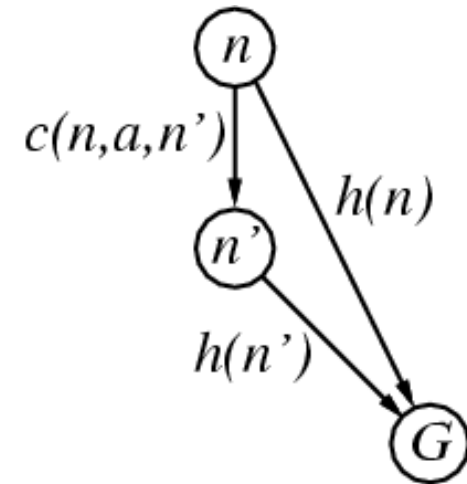
- Una heurística es **consistente** si para cada nodo n , todo sucesor n' de n generado por cualquier acción a , cumple que:

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

- Si h es consistente, tenemos que

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

- Es decir, $f(n)$ es no decreciente a lo largo de cualquier camino



Heurísticas consistentes

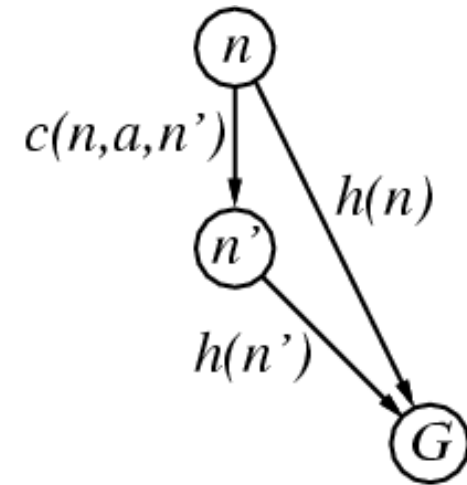
- Una heurística es **consistente** si para cada nodo n , todo sucesor n' de n generado por cualquier acción a , cumple que:

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

- Si h es consistente, tenemos que

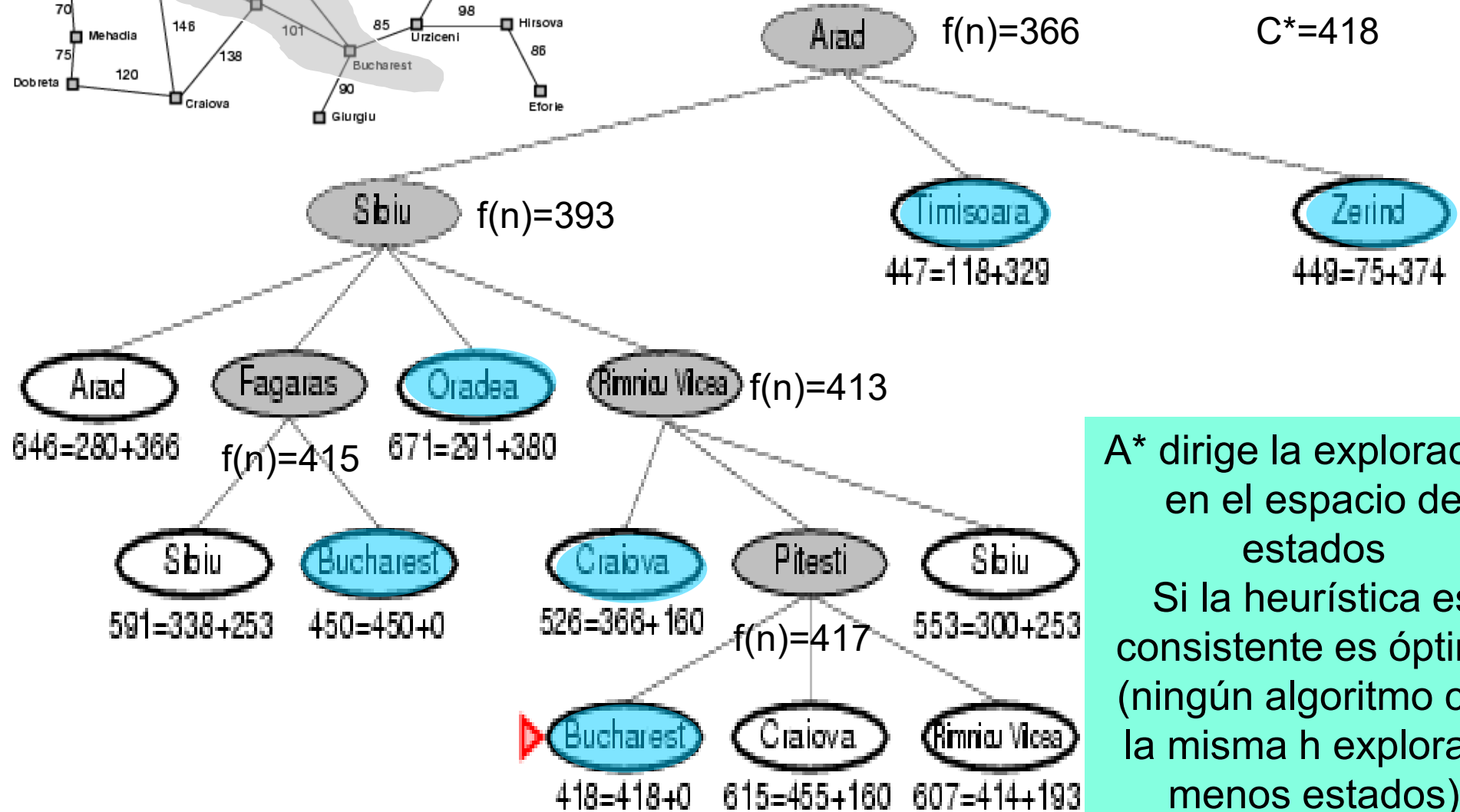
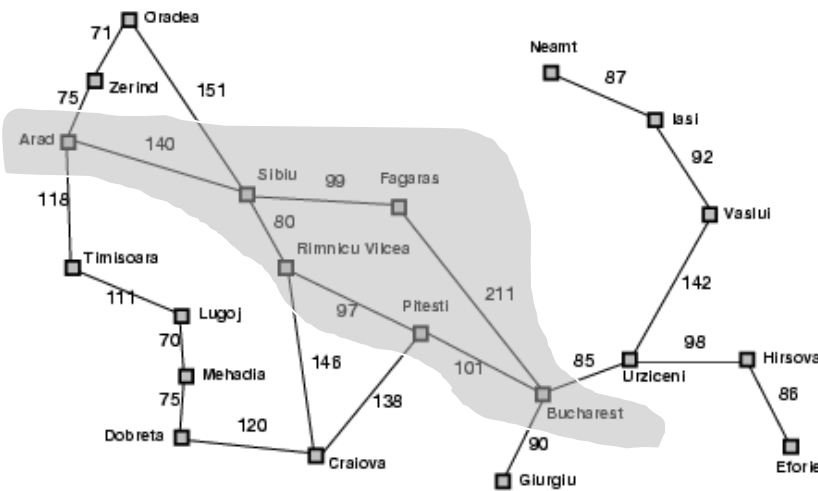
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

- Es decir, $f(n)$ es no decreciente a lo largo de cualquier camino



Teorema: Si $h(n)$ es **consistente**, la búsqueda **A*** utilizando **búsqueda en grafos** es **óptima**

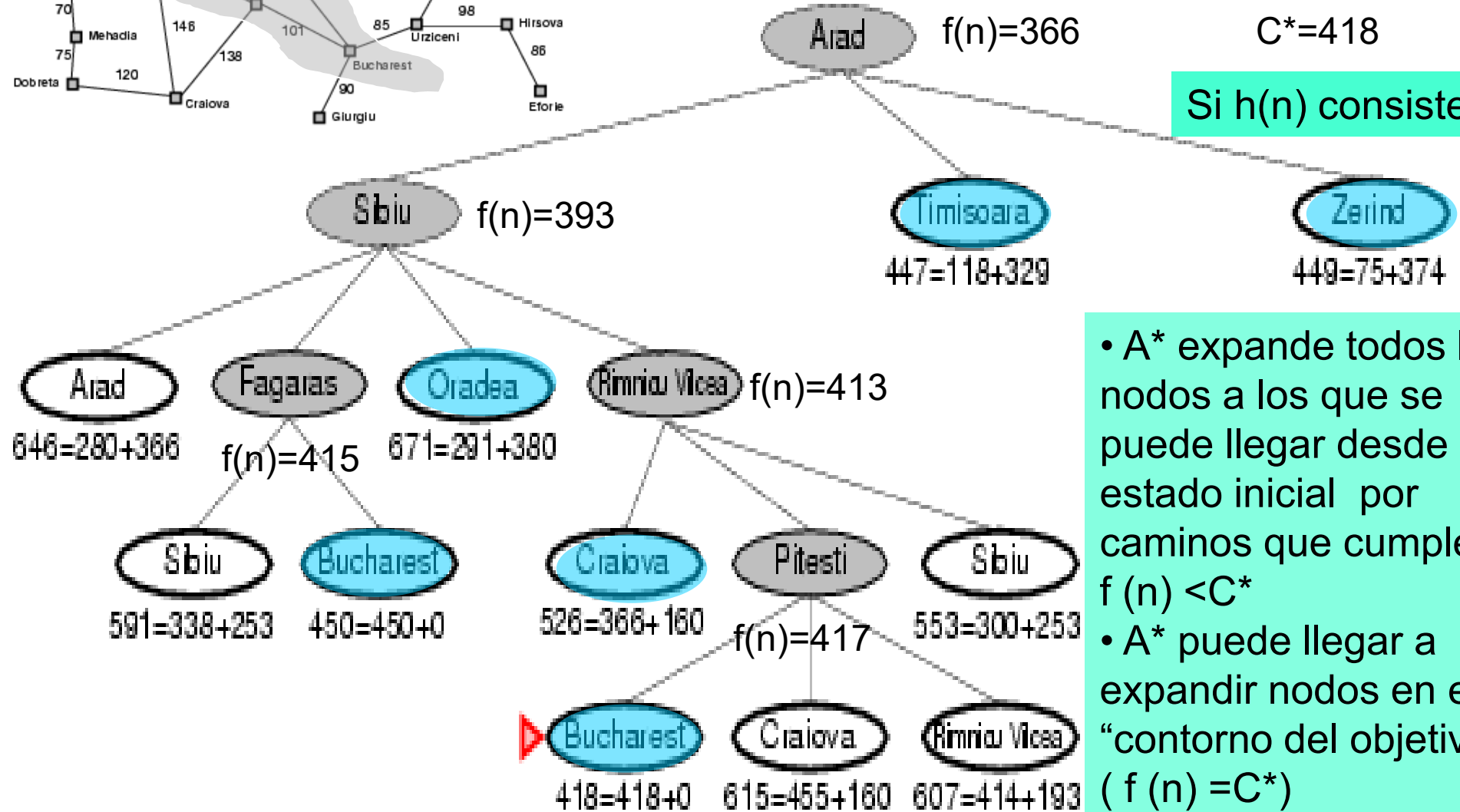
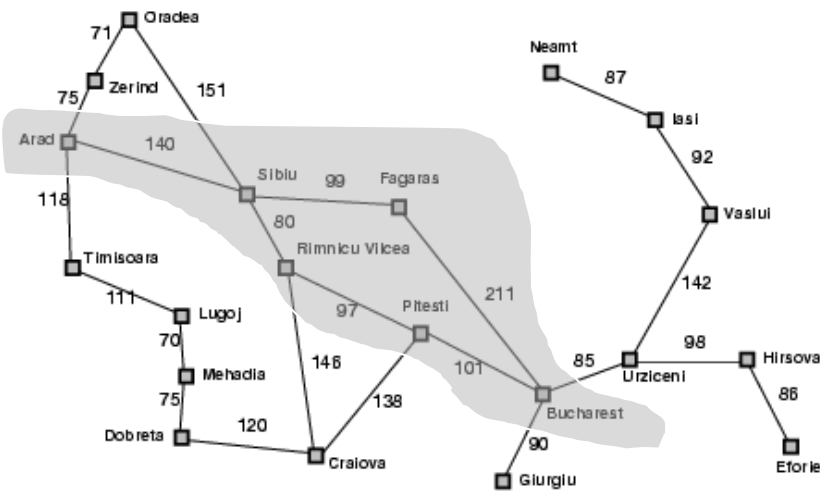
la búsqueda A*



A* dirige la exploración en el espacio de estados
Si la heurística es consistente es óptimo (ningún algoritmo con la misma h explorará menos estados)

la búsqueda A*

C^* : coste del camino óptimo



- A* expande todos los nodos a los que se puede llegar desde el estado inicial por caminos que cumplen: $f(n) < C^*$
- A* puede llegar a expandir nodos en el “contorno del objetivo” ($f(n) = C^*$)
- A* nunca expande nodos con $f(n) > C^*$.

Heurísticas admisibles

Para el 8-puzzle?

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Heurísticas admisibles

Para el 8-puzzle:

- $h_1(n)$ = número de cuadrados fuera de sitio
- $h_2(n)$ = distancia total de Manhattan (es decir, número de movimientos desde la posición actual hasta la posición objetivo para cada cuadrado)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$h_1(S) = ?$

$h_2(S) = ?$

Heurísticas admisibles

Para el 8-puzzle:

- $h_1(n)$ = número de cuadrados fuera de sitio
- $h_2(n)$ = distancia total de Manhattan (es decir, número de movimientos desde la posición actual hasta la posición objetivo para cada cuadrado)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

$$h_1(S) = ? \ 8$$

$$h_2(S) = ? \ 3+1+2+2+3+2+2+3= 18$$

7 2 4 5 6 8 3 1

Dominancia

- Si $h_2(n) \geq h_1(n)$ para todo n (siendo ambas admisibles) entonces h_2 **domina** h_1
- h_2 es mejor para buscar

Dominancia

- Si $h_2(n) \geq h_1(n)$ para todo n (siendo ambas admisibles) entonces h_2 **domina** h_1
- h_2 es mejor para buscar
- Costes típicos de búsqueda (número medio de nodos generados):

8-puzzle: (d: depth (profundidad de la solución))

- $d=10$ $BFS = 1033$ nodos
 $A^*(h_1) = 116$ nodos
 $A^*(h_2) = 48$ nodos

Dominancia

- Si $h_2(n) \geq h_1(n)$ para todo n (siendo ambas admisibles) entonces h_2 **domina** h_1
- h_2 es mejor para buscar
- Costes típicos de búsqueda (número medio de nodos generados):

8-puzzle: (d: depth (profundidad de la solución))

- $d=10$ $BFS = 1033$ nodos
 $A^*(h_1) = 116$ nodos
 $A^*(h_2) = 48$ nodos
- $d=20$ $BFS = 91493$ nodos
 $A^*(h_1) = 9905$ nodos
 $A^*(h_2) = 1318$ nodos
- $d=28$ $BFS = 463234$ nodos
 $A^*(h_1) = 202565$ nodos
 $A^*(h_2) = 22055$ nodos

Problemas relajados

- Un problema con menos restricciones en las acciones se denomina **problema relajado**.
- El coste de una solución óptima a un problema relajado es una heurística admisible del problema original.

Problemas relajados

- Un problema con menos restricciones en las acciones se denomina **problema relajado**.
- El coste de una solución óptima a un problema relajado es una heurística admisible del problema original.
- Si las reglas del 8-puzzle se relajan de forma que un cuadrado se pueda mover a **cualquier parte**, entonces $h_1(n)$ proporciona la solución más corta.
- Si las reglas se relajan de forma que un cuadrado se puede mover a **cualquier cuadrado adyacente**, entonces $h_2(n)$ nos da la solución más corta.

Resumen (hasta ahora)

- Las heurísticas nos permiten introducir información particular del problema para guiar la búsqueda
- Si únicamente utilizamos la información de cuánto nos queda para llegar → primero mejor greedy
- Si unimos a lo que nos queda para llegar cuánto llevamos recorrido → A^*
- A^* es óptimo con heurísticas admisibles (y en grafos con heurísticas consistentes)
- El diseño de buenas heurísticas es clave. Se pueden utilizar problemas relajados.