



UNIVERSIDAD
DE GRANADA



Técnicas de los Sistemas Inteligentes

Grado en Informática

Curso 2021-22. Práctica 1 Algoritmos BFS, DFS y A*

Jesús Giráldez Crú y Pablo Mesejo Santiago

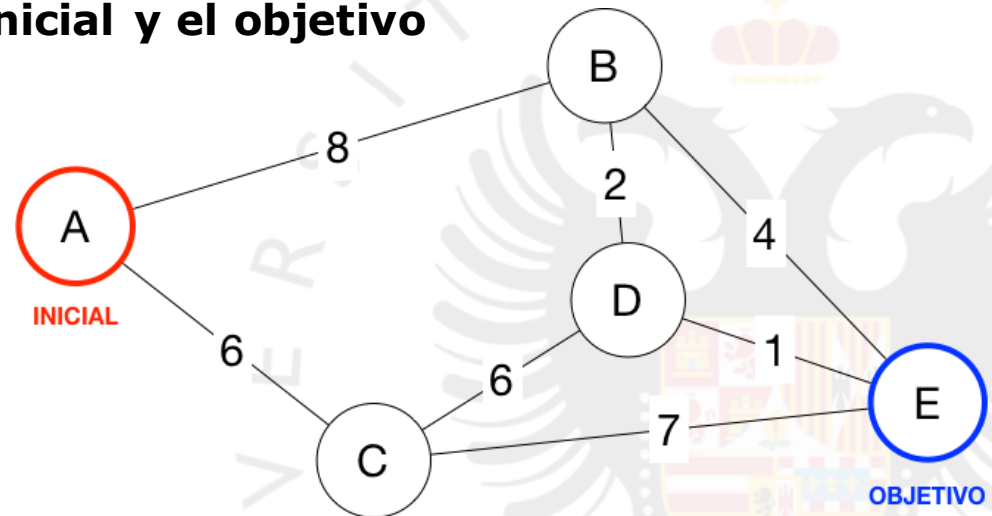
Departamento de Ciencias de la
Computación e Inteligencia Artificial
<http://decsai.ugr.es>

- Input:**

- Un grafo
- Un nodo inicial
- Un nodo objetivo
- Una función heurística (sólo en búsqueda heurística)

- Output:**

- Una ruta entre el nodo inicial y el objetivo



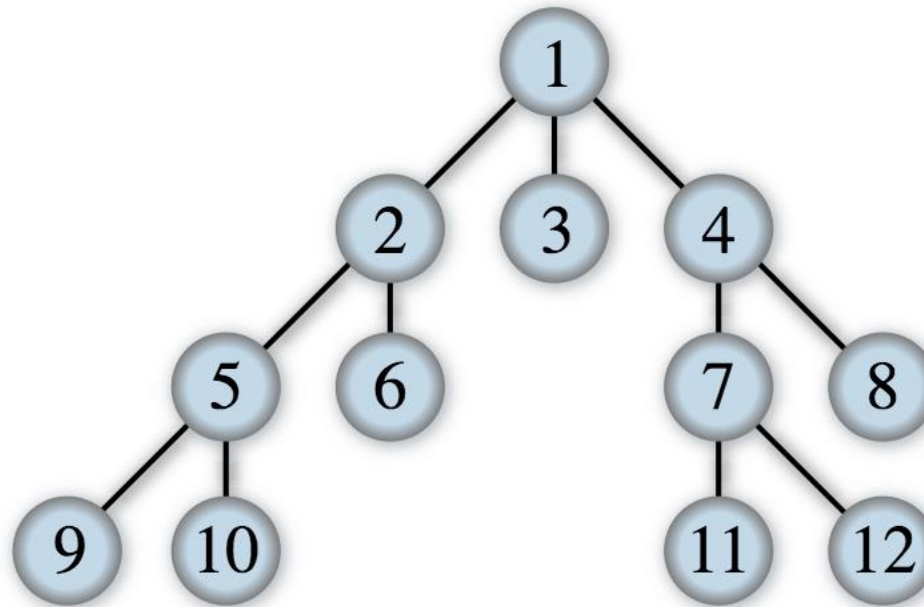
	A	B	C	D	E
$h(n)$	10	4	5	1	0

Búsqueda no informada:

BFS y DFS



- Encuentra el camino óptimo (menor profundidad) cuando el coste de desplazamiento entre nodos es constante
 - Si el coste es variable, no se garantiza el óptimo
- Es muy ineficiente en tiempo y memoria
 - Tiempo: expande muchas ramas que no son útiles
 - Memoria: mantiene todo el árbol expandido



[S.J. Russel, P. Norvig. Artificial Intelligence: A modern approach. Pearson, 2020]

```

BFS(Nodo inicial, Nodo objetivo)
  estado[inicial] = VISITADO
  padre[inicial] = null
  Cola Q = [inicial]
  while ! Q.vacía()

```

```

    u = Q.extraer()

```

```

    if u==objetivo FIN

```

```

    for each v in sucesores(u)

```

```

        if estado[v] == NOVISITADO

```

```

            estado[v] = VISITADO

```

```

            padre[v] = u

```

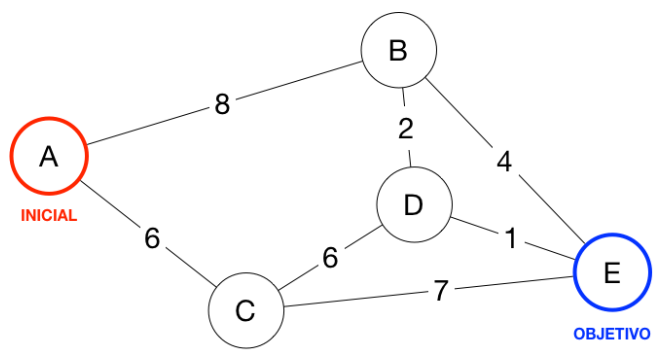
```

            Q.encolar(v)

```

u es un nodo
expandido!

Una sola cola y un estado asociado a cada nodo:
más eficiente que usar dos colas (Abierto y Cerrados)



	A	B	C	D	E
h(n)	10	4	5	1	0

It.	Nodo expandido (nodo padre)	Cola	estado				
			A	B	C	D	E
0	-	[A]	V	NV	NV	NV	NV
1	A (-)	[B,C]	V	V	V	NV	NV
2	B (A)	[C,D,E]	V	V	V	V	V
3	C (A)	[D,E]	V	V	V	V	V
4	D (B)	[E]	V	V	V	V	V
5	E (B)	Fin. Camino encontrado: E – B – A					

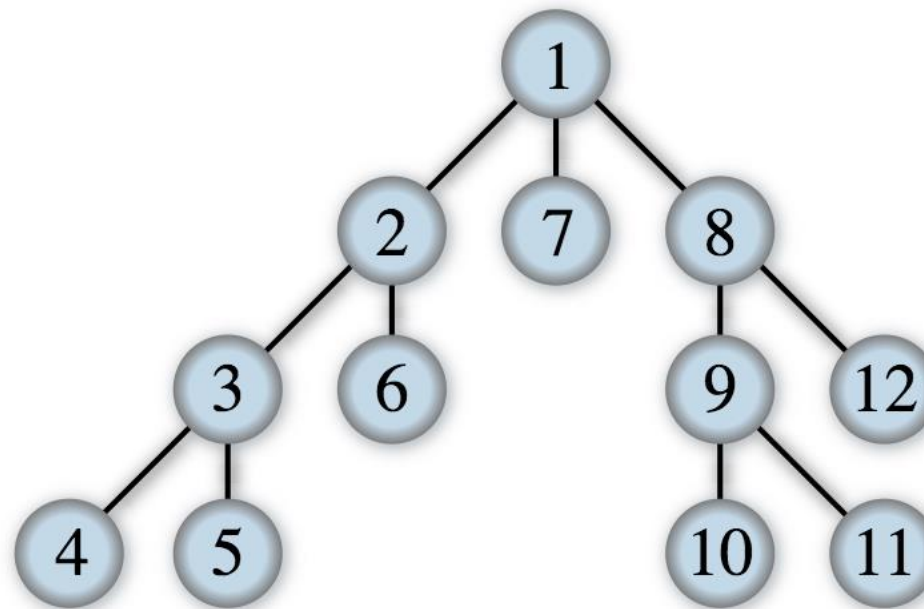
V: visitado / NV: no visitado

Orden de expansión: orden alfabético (A,B,C, ...)



No es óptimo!

- No garantiza encontrar el camino óptimo
 - Incluso si el coste de desplazamiento entre nodos es constante
- En el caso medio es más eficiente que BFS en tiempo
 - Sólo en el peor caso ambos algoritmos tardarán lo mismo
- Es más eficiente que BFS en memoria
 - Sólo requiere mantener en memoria la rama que se está explorando

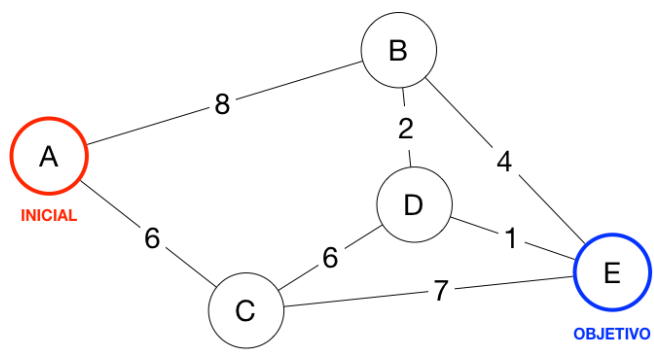


[S.J. Russel, P. Norvig. Artificial Intelligence: A modern approach. Pearson, 2020]

```
DFS(Nodo inicial, Nodo objetivo)
    estado[inicial] = VISITADO
    padre[inicial] = null
    DFS_search(inicial, objetivo)
```

```
DFS_search(Nodo u, Nodo objetivo)
    if u==objetivo return TRUE
    for each v in sucesores(u)
        if estado[v] == NOVISITADO
            estado[v] = VISITADO
            padre[v] = u
            return DFS(v, objetivo)
    return FALSE
```

u es un nodo
expandido!



	A	B	C	D	E
$h(n)$	10	4	5	1	0

It.	Nodo expandido	Sucesores no visitados	Rama actual	Otras ramas pendientes de explorar (si la actual falla)
0	-	-	[A]	-
1	A	B,C	[A,B]	[A,C]
2	B	D,E	[A,B,D]	[A,B,E], [A,C]
3	D	C,E	[A,B,D,C]	[A,B,D,E], [A,B,E], [A,C]
4	C	E	[A,B,D,C,E]	[A,B,D,E], [A,B,E], [A,C]
5	E	Fin.	Camino encontrado: E – C – D – B – A	

Orden de expansión de sucesores: orden alfabético (A,B,C, ...)



No es óptimo!

- Otros algoritmos de búsqueda no informada (**NO usados** en esta práctica):
- Algoritmo de Dijkstra (también llamado “coste uniforme”)
 - Garantiza el óptimo incluso si el coste de desplazamientos es variable
 - Intuitivamente sigue una estrategia en “anchura”
 - Pero la cola de nodos pendientes de expansión se ordena por $g(n)$
- Profundidad iterativa (IDS)
 - Intuitivamente sigue una estrategia en “profundidad”
 - Menor consumo de memoria
 - También garantiza el camino óptimo al utilizar una profundidad acotada que iterativamente se va incrementando
- Ninguno utiliza información heurística
 - Las variantes heurísticas más similares son A^* e IDA^*

Dijkstra: [Dijkstra. 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1: 269-271]

IDS: [Korf. 1985. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence. 27: 97–109]

Búsqueda heurística:

A^* e IDA^*



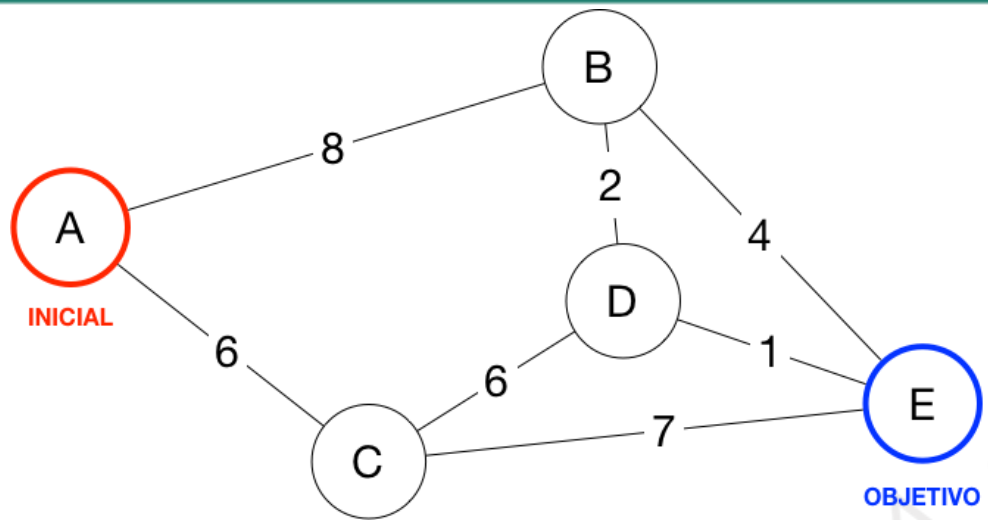
- Ambos encuentran el camino óptimo
- Son algoritmos de búsqueda heurística offline: la búsqueda completa se realiza ANTES de la ejecución
- IDA* es una extensión de DFS: profundidad iterativa (lo cual asegura el camino óptimo) e información heurística (para el orden de expansión de los vecinos de un nodo)
- Comparativa entre ambos:
 - A* consume más memoria que IDA*
 - A* es más rápido que IDA*

A*: [Hart, Nilsson, Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics. 4 (2): 100–107]

IDA*: [Korf. 1985. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence. 27: 97–109]

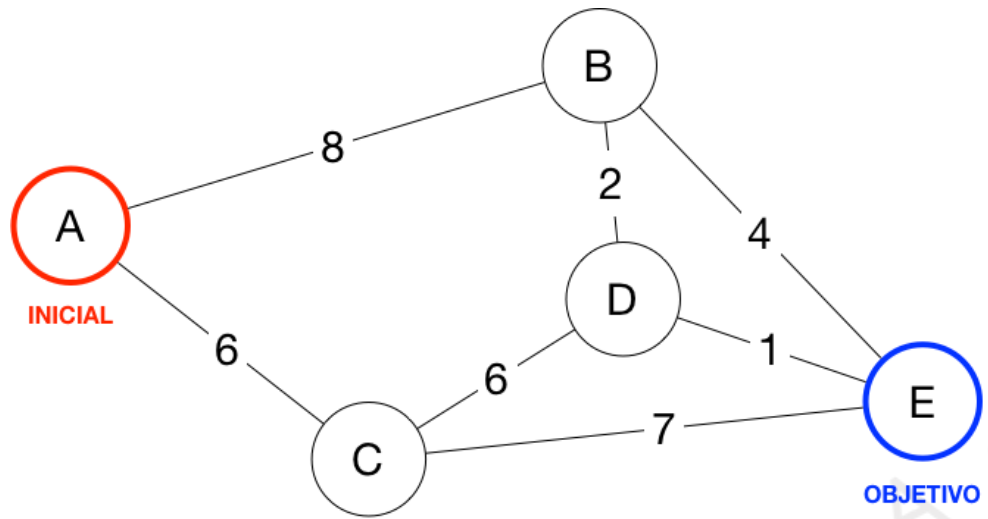
- *abiertos* = [*inicial*] # **nodos pendientes de visitar**
- *cerrados* = [] # **nodos ya visitados**
- **while True** :
 - *nodo* = **mejorCandidato**(*abiertos*) # **mejor f(n)**
 - **if** *nodo* == *objetivo*:
 - **break** # **se expande el nodo objetivo, fin.**
 - *abiertos.remove(nodo)*
 - **foreach** *sucesor* in **expandir**(*nodo*) : # **vecinos del nodo**
 - **if** *sucesor* == *nodo.parent* :
 - **Pass** # **nodo padre ya visitado**
 - **if** *cerrados.contains(sucesor)*
 - **and** **mejorCaminoA**(*sucesor*): # **menor g(n)**
 - *cerrados.remove(sucesor)*
 - *abiertos.add(sucesor)* # **actualizar g(n)**
 - **elif not** *cerrados.contains(sucesor)* **and** **not** *abiertos.contains(sucesor)* :
 - *abiertos.add(sucesor)* # **nodo no visitado, añadir**
 - **elif** *abiertos.contains(sucesor)* **and** **mejorCaminoA**(*sucesor*):
 - *abiertos.update(sucesor)* # **actualizar g(n)**

nodo es un
nodo
expandido!



	A	B	C	D	E
h(n)	10	4	5	1	0

It.	Expandir	Sucesores	Abiertos	Cerrados
0			A(0+10)	-
1	A	B(8+4), C(6+5)	B(8+4), C(6+5)	A
2	C	D(12+1), E(13+0)	B(8+4), D(12+1), E(13+0)	A,C
3	B	D(10+1), E(12+0)	D(12+1), E(13+0) , D(10+1), E(12+0)	A,C,B
4	D	C(16+5) , E(11+0)	E(12+0) , E(11+0)	A,C,B,D
5	E	Fin		A,C,B,D,E

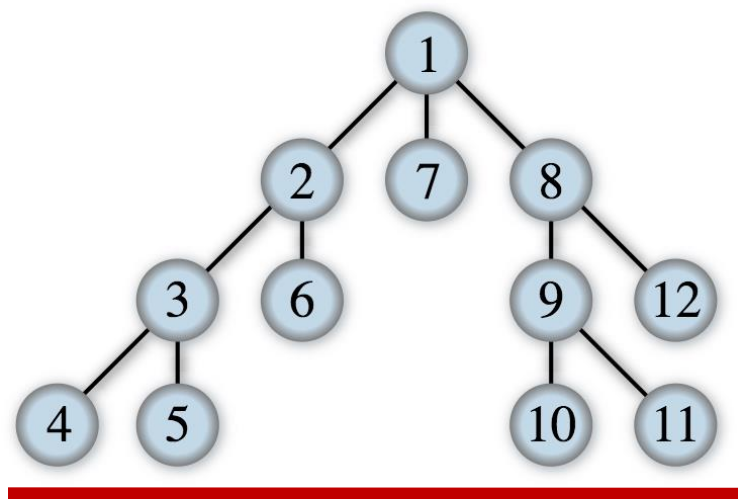
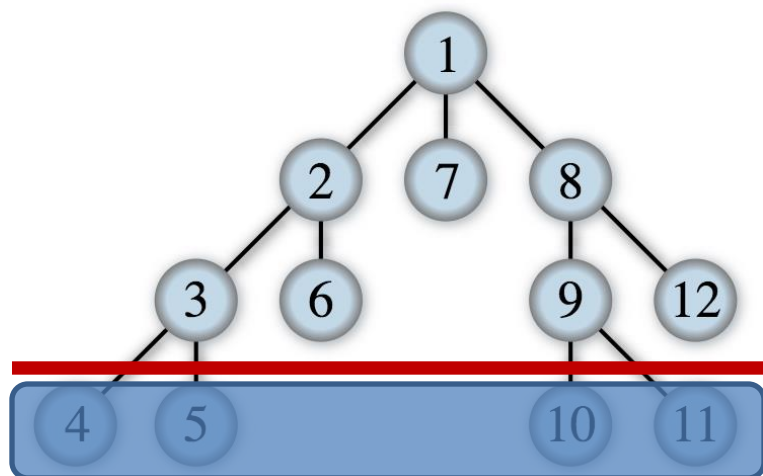
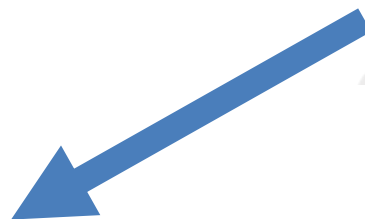
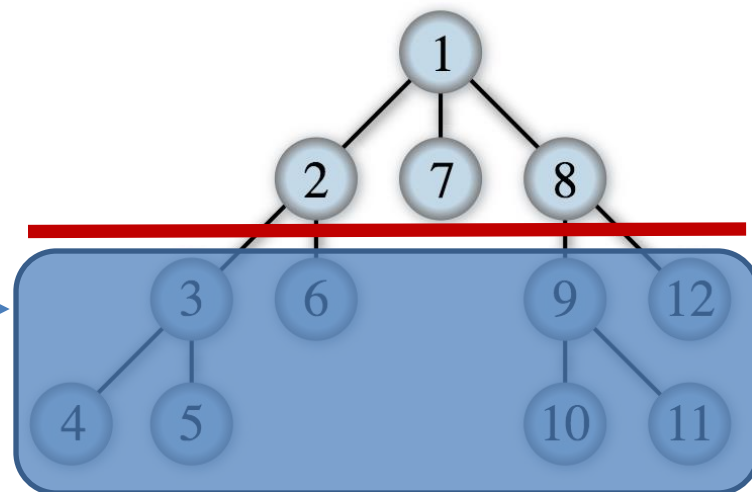
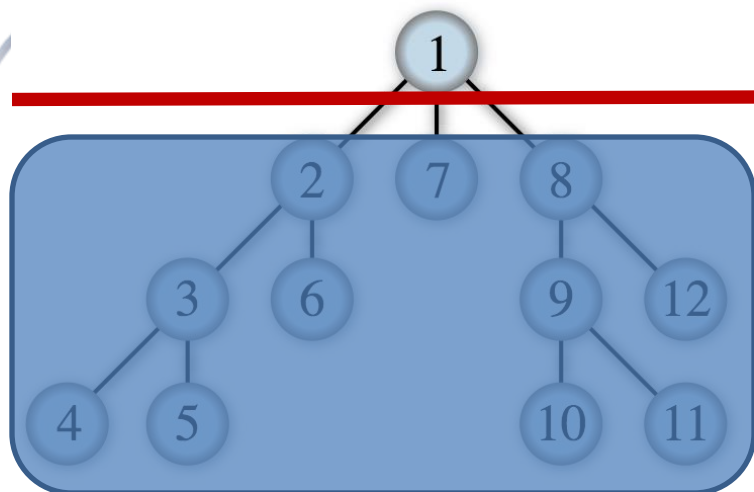


	A	B	C	D	E
h(n)	10	4	5	1	0

Nodo	Nodo padre	Coste del arco	Ruta	Coste de la ruta
E (Objetivo)	D	1	E - D	1
D	B	2	E - D - B	1+2 = 3
B	A	8	E - D - B - A	1+2+8 = 11
A (Inicial)				

IDA*





```
IDAStar(Nodo inicial, Nodo objetivo)
```

```
    cota = h(inicial)
```

```
    ruta = [inicial]
```

```
    while TRUE
```

```
        t = search(ruta, 0, cota, objetivo)
```

```
        if t == ENCONTRADO return (ruta, cota)
```

```
        if t == INFINITO return NOENCONTRADO
```

```
        cota = t
```

```
search(ruta, g, cota, objetivo)
```

```
    nodo = ruta.last
```

```
    f = g + h(nodo)
```

```
    if f > cota return f
```

```
    if nodo == objetivo return ENCONTRADO
```

```
    min = INFINITO
```

```
    foreach v in sucesores(nodo).sort()
```

```
        if v not in ruta
```

```
            ruta.push(v)
```

```
            t = search(ruta, g+c(nodo,v), cota, objetivo)
```

```
            if t == ENCONTRADO return ENCONTRADO
```

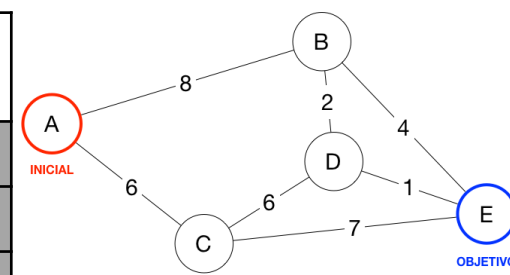
```
            if t < min min = t
```

```
            ruta.pop()
```

```
return min
```

nodo es un
nodo
expandido!

Ordenados por
 $c(\text{nodo}, v)$



	A	B	C	D	E
h(n)	10	4	5	1	0

Llamadas al método search con prof. iterativa (cota)

Iter.	Cota	Nodo expandido (f=g+h)	Sucesores (g+h) ordenados	Ramas a explorar	min
0	10	-	-	[A]	-
1	10	A (10=0+10)	C(6+5), B(8+4)	[A,C], [A,B]	INF
2	10	C (11=6+5)	Stop (f > cota)	[A,B]	11
3	10	B (12=8+4)	Stop (f > cota)	-	11
4	11	-	-	[A]	-
5	11	A (10=0+10)	C(6+5), B(8+4)	[A,C], [A,B]	INF
6	11	C (11=6+5)	D(12+1), E(13+0)	[A,C,D], [A,C,E], [A,B]	INF
7	11	D (13=12+1)	Stop (f > cota)	[A,C,E], [A,B]	13
8	11	E (13=13+0)	Stop (f > cota)	[A,B]	13
9	11	B (12=8+4)	Stop (f > cota)	-	12
10	12	-	-	[A]	-
11	12	A (10=0+10)	C(6+5), B(8+4)	[A,C], [A,B]	INF
12	12	C (11=6+5)	D(12+1), E(13+0)	[A,C,D], [A,C,E], [A,B]	INF
13	12	D (13=12+1)	Stop (f > cota)	[A,C,E], [A,B]	13
14	12	E (13=13+0)	Stop (f > cota)	[A,B]	13
15	12	B (12=8+4)	D(10+1), E(12+0)	[A,B,D], [A,B,E]	13
16	12	D(11=10+1)	E(11+0)	[A,B,D,E], [A,B,E]	13
17	12	E(11=11+0)	FIN. Camino: E – D – B – A		ENC

Para más información:

Introduction to the A* algorithm:

<https://www.redblobgames.com/pathfinding/a-star/introduction.html>

Aplicación visual para evaluar distintos algoritmos en grids:

<https://qiao.github.io/PathFinding.js/visual/>