

FUNDAMENTOS DE SISTEMAS  
INTELIGENTES

ESTRATEGIAS DE  
**BUSQUEDA**

JUAN CARLOS DOMÍNGUEZ DOPAZO  
— ASMAE EZ ZAIM DRIOUCH —

POINTS

# INTRODUCCIÓN

- **Introducción:** En esta presentación, exploraremos la implementación del algoritmo de Ramificación y Acotación y sus mejoras significativas.
- **Objetivo de la Presentación:** Comprender cómo el enfoque inicial evolucionó hacia una solución más eficiente que considera la subestimación.



# PRIMERA IMPLEMENTACIÓN



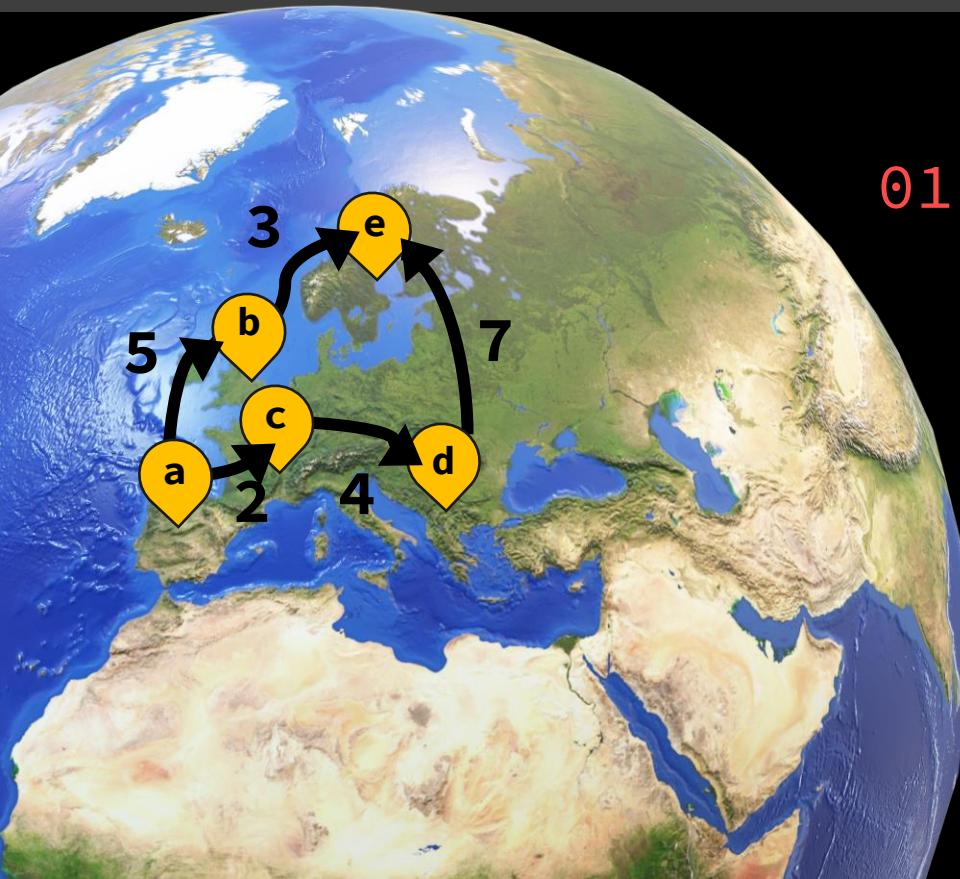
# PRIMERA IMPLEMENTACIÓN



## 01 Objetivo

Implementar un método de ramificación y acotación que busque siempre el camino más corto de un nodo a otro

# PRIMERA IMPLEMENTACIÓN



## 01 Objetivo

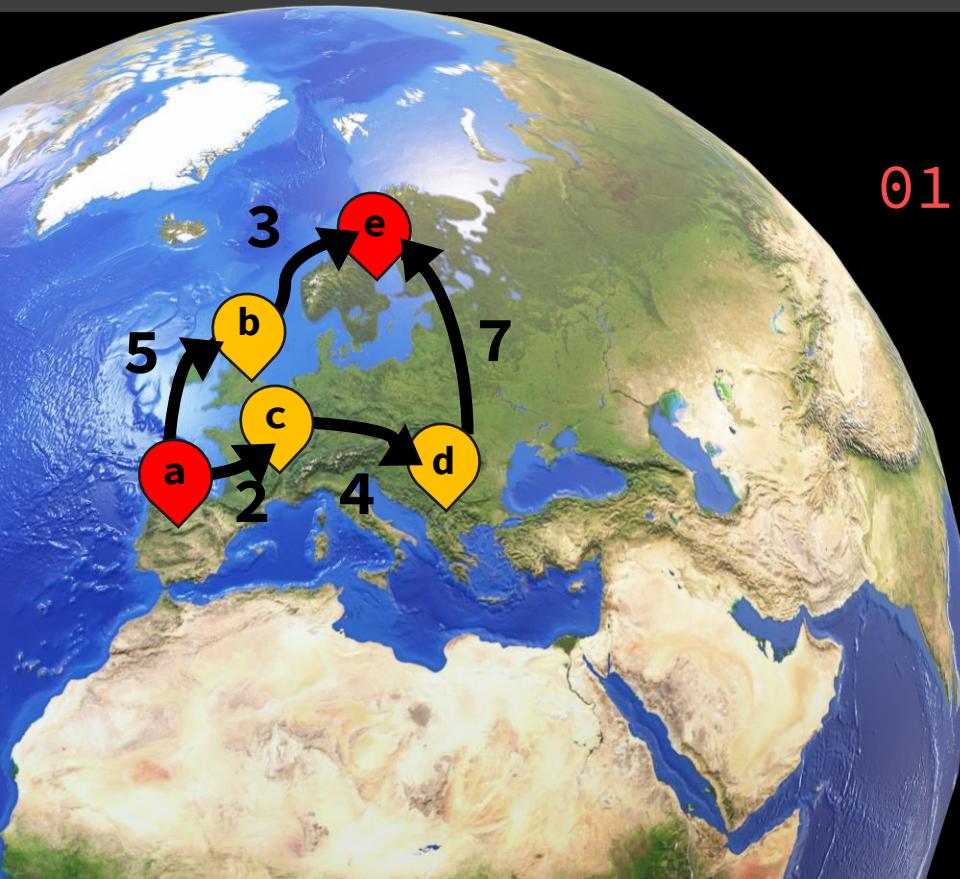
Implementar un método de ramificación y acotación que busque siempre el camino más corto de un nodo a otro

Ejemplo:

Nodo inicial = a

Nodo objetivo = e

# PRIMERA IMPLEMENTACIÓN



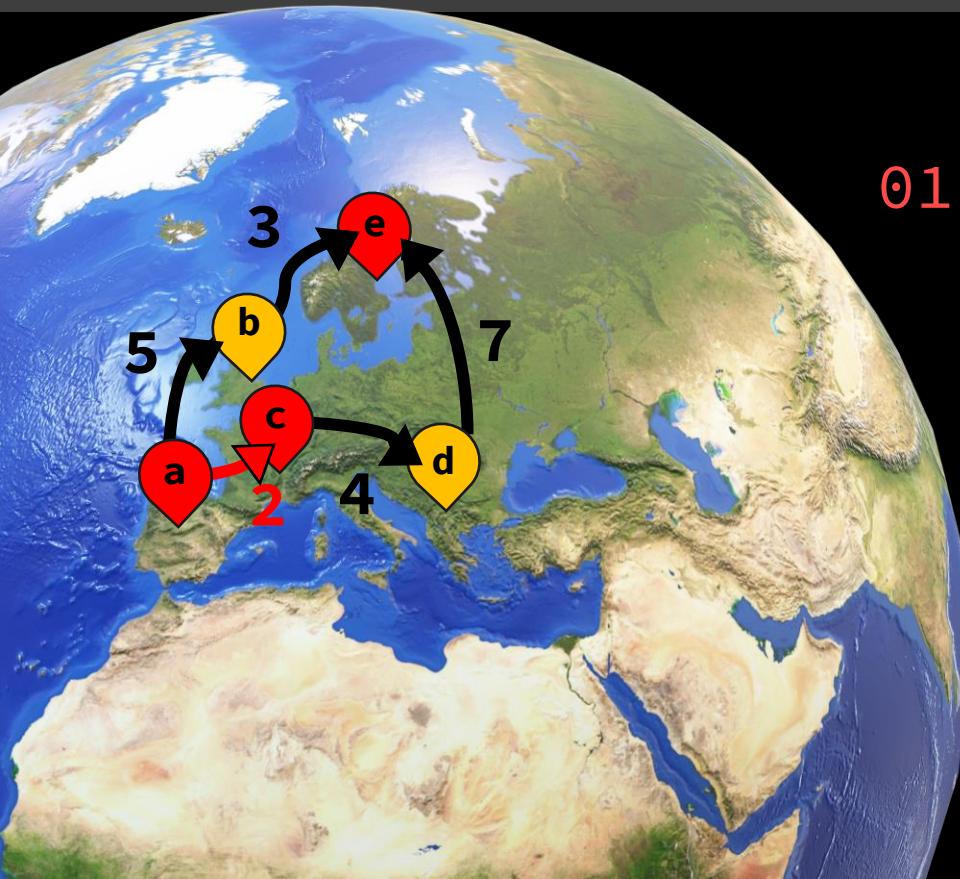
## 01 Objetivo

Ejemplo:

Nodo inicial = a  
Nodo objetivo = e

Situándonos en a, el siguiente nodo más cercano es c, ya que tiene un coste de 2.

# PRIMERA IMPLEMENTACIÓN

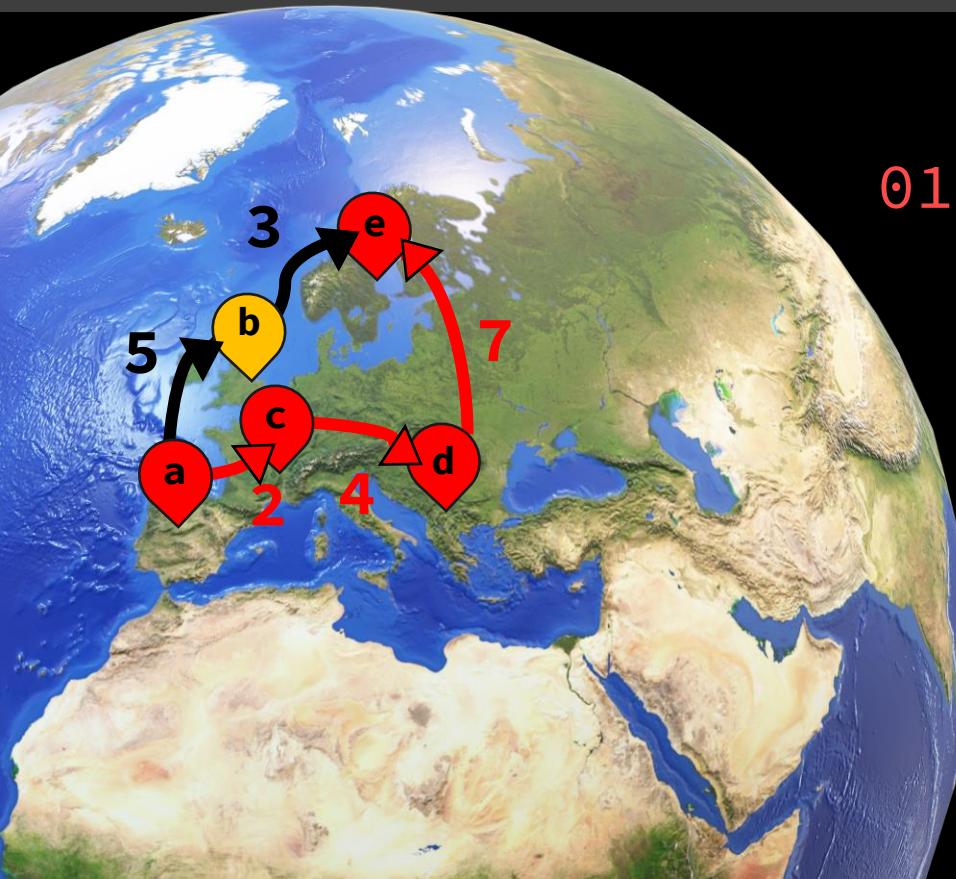


## 01 Objetivo

Situándonos en a, el siguiente nodo más cercano es c, ya que tiene un coste de 2.

Sucesivamente, se irían recorriendo los demás nodos con el mismo algoritmo. Pero al no tener en cuenta la subestimación no obtenemos el camino óptimo.

# PRIMERA IMPLEMENTACIÓN



## 01 Objetivo

Sucesivamente, se irían recorriendo los demás nodos con el mismo algoritmo. Pero al no tener en cuenta la subestimación no obtenemos el camino óptimo.

Si el árbol es muy grande el procedimiento va a ser muy ineficiente, puesto que no hace backtracking

---

# MODIFICACIÓN DEL CÓDIGO

---



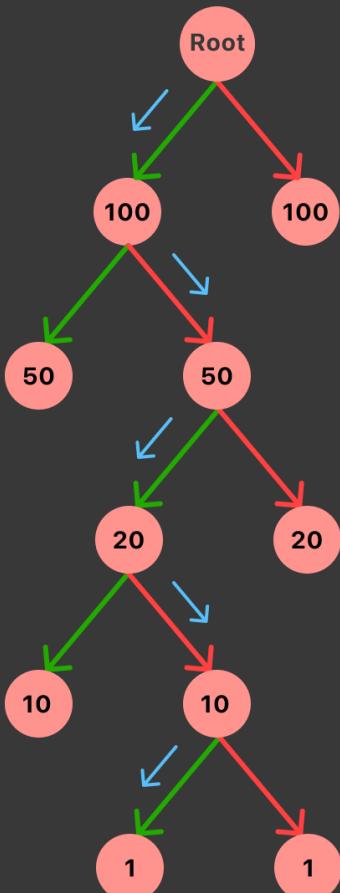
```
97 def graph_search(problem, fringe):
98     """Search through the successors of a problem to find a goal.
99     The argument fringe should be an empty queue.
100    If two paths reach a state, only use the best one. [Fig. 3.18]"""
101    closed = {}
102
103    generatedNodes = 1 # Suponer que el grafo nunca está vacío
104    visitedNodes = 0
105
106    start_time = time.perf_counter() * 1000 # Registro del tiempo de inicio en milisegundos
107
108    fringe.append(Node(problem.initial))
109
110    while fringe:
111        node = fringe.pop()
112        # Se incrementa el marcador de nodos visitados
113        visitedNodes += 1
114        if problem.goal_test(node.state):
115            end_time = time.perf_counter() * 1000 # Registro del tiempo de finalización en milisegundos
116            elapsed_time = end_time - start_time # Calcula el tiempo transcurrido en milisegundos
117            r = f"Generados: {generatedNodes}\nVisitados: {visitedNodes}\nCosto total: {node.path_cost}\nTiempo total: {elapsed_time} (ms)"
118            print(r)
119            return node
120        if node.state not in closed:
121            closed[node.state] = True
122            successors = node.expand(problem)
123            fringe.extend(successors)
124            # Incrementar nodos generados
125            generatedNodes += len(successors)
126
127    return None
```



```
97 def graph_search(problem, fringe):
98     """Search through the successors of a problem to find a goal.
99     The argument fringe should be an empty queue.
100    If two paths reach a state, only use the best one. [Fig. 3.18]"""
101    closed = {}
102
103    generatedNodes = 1 # Suponer que el grafo nunca está vacío
104    visitedNodes = 0
105
106    start_time = time.perf_counter() * 1000 # Registro del tiempo de inicio en milisegundos
107
108    fringe.append(Node(problem.initial))
109
110    while fringe:
111        node = fringe.pop()
112        # Se incrementa el marcador de nodos visitados
113        visitedNodes += 1
114        if problem.goal_test(node.state):
115            end_time = time.perf_counter() * 1000 # Registro del tiempo de finalización en milisegundos
116            elapsed_time = end_time - start_time # Calcula el tiempo transcurrido en milisegundos
117            r = f"Generados: {generatedNodes}\nVisitados: {visitedNodes}\nCosto total: {node.path_cost}\nTiempo total: {elapsed_time} (ms)"
118            print(r)
119            return node
120        if node.state not in closed:
121            closed[node.state] = True
122            successors = node.expand(problem)
123            fringe.extend(successors)
124            # Incrementar nodos generados
125            generatedNodes += len(successors)
126
127    return None
```



```
97 def graph_search(problem, fringe):
98     """Search through the successors of a problem to find a goal.
99     The argument fringe should be an empty queue.
100    If two paths reach a state, only use the best one. [Fig. 3.18]"""
101    closed = {}
102
103    generatedNodes = 1 # Suponer que el grafo nunca está vacío
104    visitedNodes = 0
105
106    start_time = time.perf_counter() * 1000 # Registro del tiempo de inicio en milisegundos
107
108    fringe.append(Node(problem.initial))
109
110    while fringe:
111        node = fringe.pop()
112        # Se incrementa el marcador de nodos visitados
113        visitedNodes += 1
114        if problem.goal_test(node.state):
115            end_time = time.perf_counter() * 1000 # Registro del tiempo de finalización en milisegundos
116            elapsed_time = end_time - start_time # Calcula el tiempo transcurrido en milisegundos
117            r = f"Generados: {generatedNodes}\nVisitados: {visitedNodes}\nCosto total: {node.path_cost}\nTiempo total: {elapsed_time} (ms)"
118            print(r)
119            return node
120        if node.state not in closed:
121            closed[node.state] = True
122            successors = node.expand(problem)
123            fringe.extend(successors)
124            # Incrementar nodos generados
125            generatedNodes += len(successors)
126
127    return None
```



# 02 BRANCH AND BOUND

En esta etapa, veremos cómo se diseñó un algoritmo mejorado que incorpora la subestimación en su criterio de selección.

# BRANCH AND BOUND

```
138 # Branch and bound  
139 def branch_and_bound_graph_search(problem):  
140     return graph_search(problem, BnB());  
141
```

Todos los algoritmos de búsqueda utilizan la misma función, variando la estructura de datos que emplean para cada caso. En el caso del Branch and bound, se crea una clase que se llama BnB. Para el b&b con subestimación es prácticamente igual, pero a BnB\_sub() le pasamos el problem.



```
548 class BnB(Queue):
549     """ Queue that implements fundamentals of Branch and Bounding """
550     def __init__(self):
551         self.A = []
552         self.start = 0
553
554     def append(self, item):
555         self.A.append(item)
556
557     def __len__(self):
558         return len(self.A) - self.start
559
560     def extend(self, items):
561         self.A.extend(items)
562         # Ordenado de menor a mayor coste, según la longitud del camino (problema)
563         # i = problema ;; problema.path_cost
564         # Esto hace que elijamos siempre el nodo con el camino más corto == branch
565         # Ya que al extender, añadimos un nodo, y al añadir un nodo a la lista los
566         self.A = sorted(self.A, key = lambda i: i.path_cost, reverse=False)
567
568     def pop(self):
569         e = self.A[self.start]
570         self.start += 1
571         if self.start > 5 and self.start > len(self.A) / 2:
572             self.A = self.A[self.start:]
573             self.start = 0
574
575         return e
```



## CLASE BNB

Todos los algoritmos de búsqueda utilizan la misma función, variando la estructura de datos que emplean para cada caso. En el caso del Branch and Bound, se crea una clase llamada "BnB."



```
548 class BnB(Queue):
549     """ Queue that implements fundamentals of Branch and Bounding """
550     def __init__(self):
551         self.A = []
552         self.start = 0
553
554     def append(self, item):
555         self.A.append(item)
556
557     def __len__(self):
558         return len(self.A) - self.start
559
560     def extend(self, items):
561         self.A.extend(items)
562         # Ordenado de menor a mayor coste, según la longitud del camino (problema)
563         # i = problema ;; problema.path_cost
564         # Esto hace que elijamos siempre el nodo con el camino más corto == branch
565         # Ya que al extender, añadimos un nodo, y al añadir un nodo a la lista los
566         self.A = sorted(self.A, key = lambda i: i.path_cost, reverse=False)
567
568     def pop(self):
569         e = self.A[self.start]
570         self.start += 1
571         if self.start > 5 and self.start > len(self.A) / 2:
572             self.A = self.A[self.start:]
573             self.start = 0
574
575         return e
```



La particularidad de esta clase se encuentra en la función "extend()". Dado que se busca visitar los nodos que tengan el camino más corto, hace falta organizar la estructura de datos de forma que los nodos que se van extrayendo sean los más pequeños primero, y así podar el árbol.

```
548 class BnB(Queue):
549     """ Queue that implements fundamentals of Branch and Bounding """
550     def __init__(self):
551         self.A = []
552         self.start = 0
553
554     def append(self, item):
555         self.A.append(item)
556
557     def __len__(self):
558         return len(self.A) - self.start
559
560     def extend(self, items):
561         self.A.extend(items)
562         # Ordenado de menor a mayor coste, según la longitud del camino (problema)
563         # i = problema ;; problema.path_cost
564         # Esto hace que elijamos siempre el nodo con el camino más corto == branch
565         # Ya que al extender, añadimos un nodo, y al añadir un nodo a la lista los
566         self.A = sorted(self.A, key = lambda i: i.path_cost, reverse=False)
567
568     def pop(self):
569         e = self.A[self.start]
570         self.start += 1
571         if self.start > 5 and self.start > len(self.A) / 2:
572             self.A = self.A[self.start:]
573             self.start = 0
574
575         return e
```

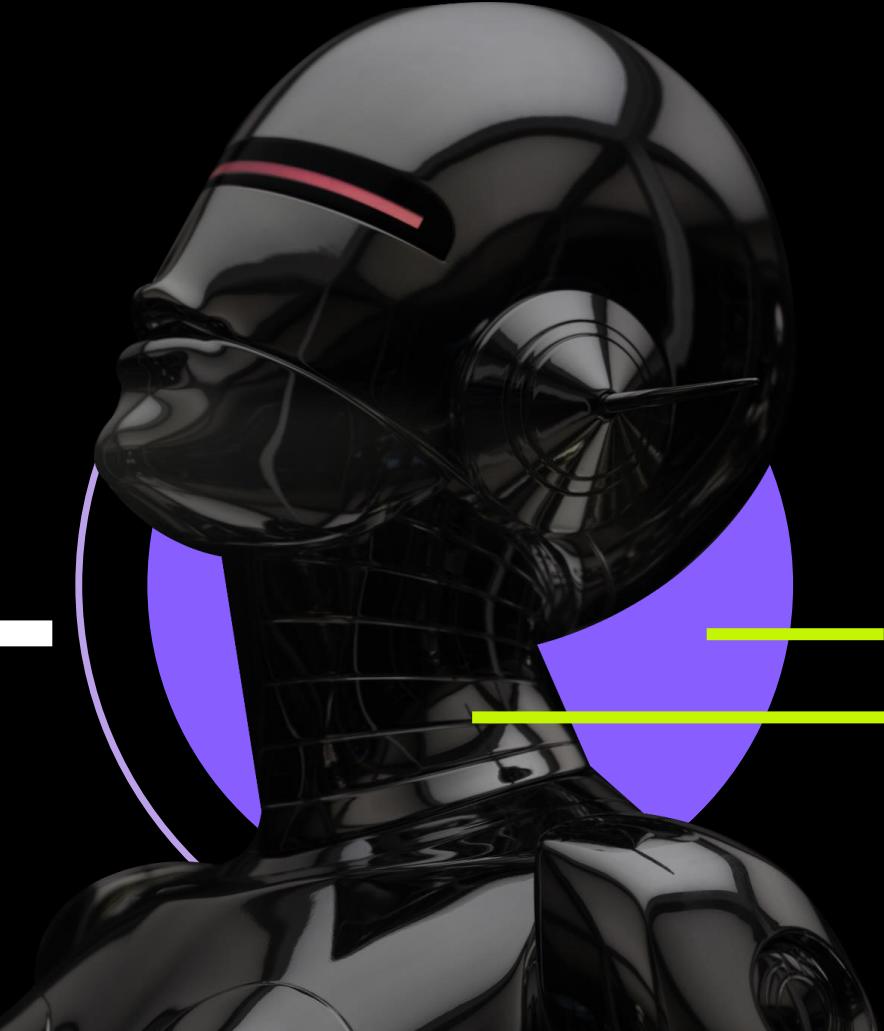


cada vez que se extiende el árbol, es decir, se añade un nuevo nodo, **se ordenan todos los elementos según el coste** del camino de manera ascendente, se consigue que cada vez que se hace un "pop()", el elemento que se extrae sea el **más pequeño**. Esto permite cumplir con el principio esencial del Branch and Bound, evitando visitar componentes que no se ajustan a la solución y llegando a la al camino más corto óptimo.

- ✓ **NO TIENE EN CUENTA LA SUBESTIMACIÓN**
- ✓ **CAMINO FINAL NO ÓPTIMO**

...

# Branch and bound con subestimación





En este nuevo enfoque, añadimos un componente adicional:



# HEURÍSTICA

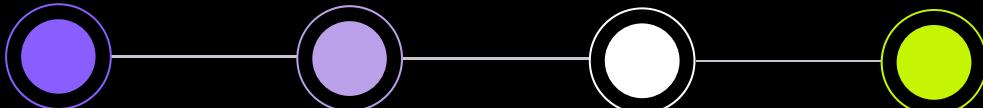
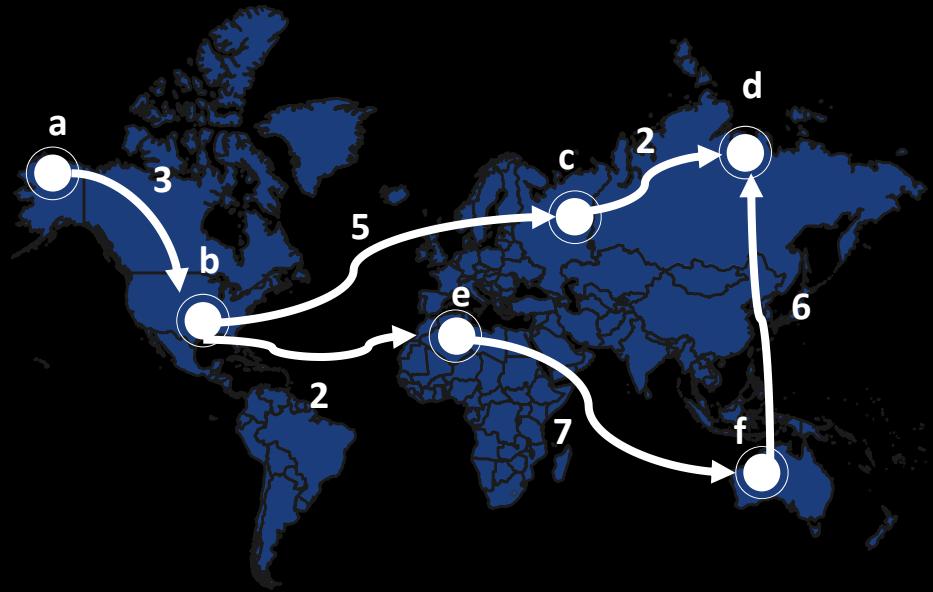
Esta heurística estima la distancia desde el nodo actual hasta el objetivo.

La heurística en este caso, es el valor de la distancia euclídea. Para garantizar que el camino encontrado sea óptimo, donde la heurística de un nodo tiene que ser menor o igual que la heurística de su vecino más cercano.

$$f^*(n) = g(n) + h^*(n)$$



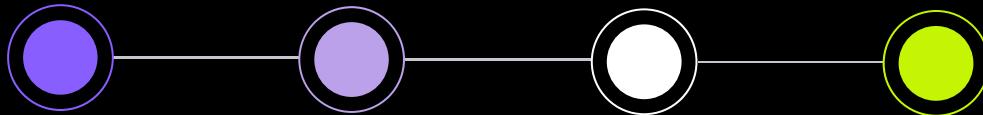
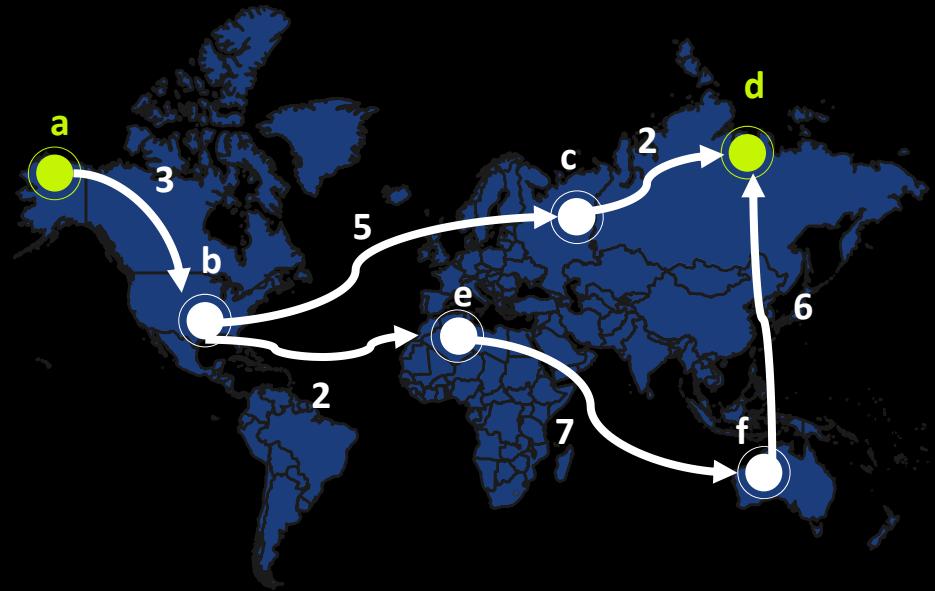
# ¿ES LA SOLUCIÓN OPTIMA?



Donde el camino elegido para llegar al destino tiene en consideración la subestimación, donde, por ejemplo, en el nodo B, se elige ir a C en vez de a E, pese a ser la distancia desde B más corta, ya que finalmente el camino total pasando por C es más corto.



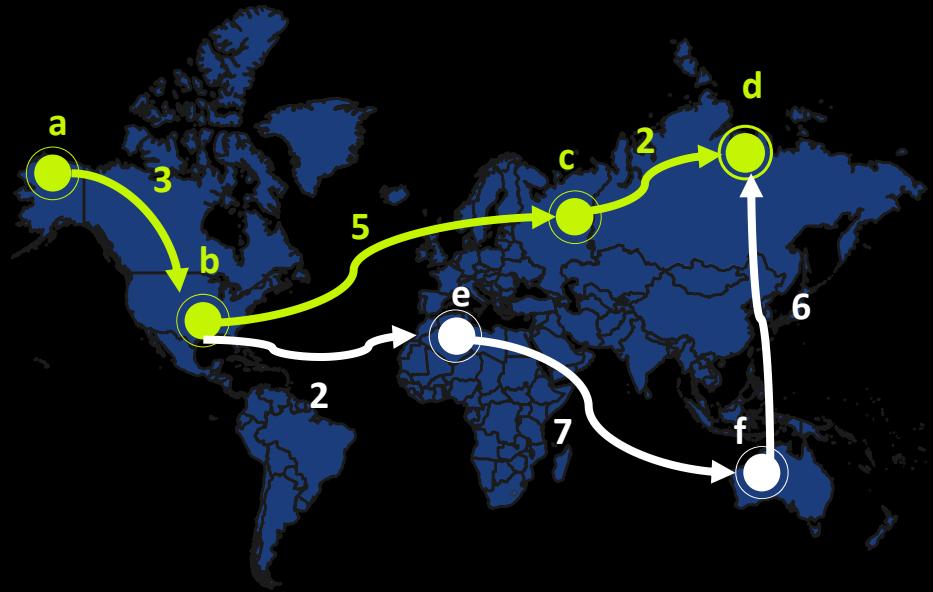
# ¿ES LA SOLUCIÓN OPTIMA?



Donde el camino elegido para llegar al destino tiene en consideración la subestimación, donde, por ejemplo, en el nodo B, se elige ir a C en vez de a E, pese a ser la distancia desde B más corta, ya que finalmente el camino total pasando por C es más corto.



# ¿ES LA SOLUCIÓN OPTIMA?



Donde el camino elegido para llegar al destino tiene en consideración la subestimación, donde, por ejemplo, en el nodo B, se elige ir a C en vez de a E, pese a ser la distancia desde B más corta, ya que finalmente el camino total pasando por C es más corto.



```
576 class BnB_sub(Queue):
577     """ Queue that implements fundamentals of Branch and Bounding with subestimation """
578     def __init__(self, sub):
579         self.A = []
580         self.start = 0
581         self.sub = sub
582
583     def append(self, item):
584         self.A.append(item)
585
586     def __len__(self):
587         return len(self.A) - self.start
588
589     def extend(self, items):
590         self.A.extend(items)
591         self.A = sorted(self.A, key = lambda i: i.path_cost + self.sub.h(i), reverse=False)
592
593     def pop(self):
594         e = self.A[self.start]
595         self.start += 1
596         if self.start > 5 and self.start > len(self.A) / 2:
597             self.A = self.A[self.start:]
598             self.start = 0
599         return e
600
601 ## Fig: The idea is we can define things like Fig[3,10] later.
602 ## Alas, it is Fig[3,10] not Fig[3.10], because that would be the same as Fig[3.1]
603 Fig = {}
```



Todos los algoritmos de búsqueda utilizan la misma función, variando la estructura de datos que emplean para cada caso. En el caso del Branch and Bound, se crea una clase llamada "BnB."

```
# Branch and bound with subestimation
def branch_and_bound_subestimation_graph_search(problem):
    return graph_search(problem, BnB_sub(problem));
```



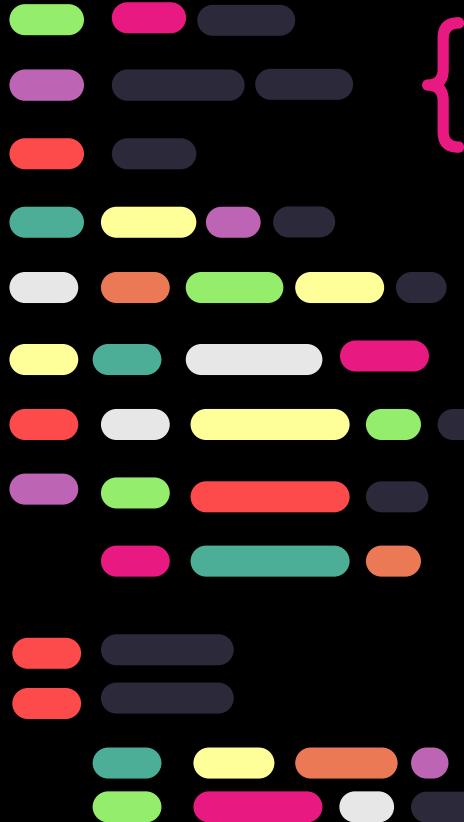
# Tabla de comparación



ID	Origen	Destino	(BFS) Amplitud	(DFS) Profundidad	Branch and Bound	Branch and Bound con subestimación
1	Arad	Bucharest	Generados: 21 Visitados: 16 Costo total: 450 Tiempo total: 0.12970000505447388 (ms) Breadth-First Path: [<Node B>, <Node F>, <Node S>, <Node A>]	Generados: 18 Visitados: 10 Costo total: 733 Tiempo total: 0.11779999732971191 (ms) Depth-First Path: [<Node B>, <Node P>, <Node C>, <Node D>, <Node M>, <Node L>, <Node T>, <Node A>]	Generados: 31 Visitados: 24 Costo total: 418 Tiempo total: 0.23079997301101685 (ms) Branch and Bound Path: [<Node B>, <Node P>, <Node R>, <Node S>, <Node A>]	Generados: 16 Visitados: 6 Costo total: 418 Tiempo total: 0.6158999800682068 (ms) Branch and Bound with subestimation Path: [<Node B>, <Node P>, <Node R>, <Node S>, <Node A>]
2	Oradea	Eforie	Generados: 45 Visitados: 43 Costo total: 730 Tiempo total: 0.27059996128082275 (ms) Breadth-First Path: [<Node E>, <Node H>, <Node U>, <Node B>, <Node F>, <Node S>, <Node O>]	Generados: 41 Visitados: 31 Costo total: 698 Tiempo total: 0.3083000183105469 (ms) Depth-First Path: [<Node E>, <Node H>, <Node U>, <Node B>, <Node P>, <Node R>, <Node S>, <Node O>]	Generados: 43 Visitados: 40 Costo total: 698 Tiempo total: 0.278499960899353 (ms) Branch and Bound Path: [<Node E>, <Node H>, <Node U>, <Node B>, <Node P>, <Node R>, <Node S>, <Node O>]	Generados: 32 Visitados: 15 Costo total: 698 Tiempo total: 0.3650999665260315 (ms) Branch and Bound with subestimation Path: [<Node E>, <Node H>, <Node U>, <Node B>, <Node P>, <Node R>, <Node S>, <Node O>]
3	Giurgiu	Zerind	Generados: 41 Visitados: 34 Costo total: 615 Tiempo total: 0.3970000147819519 (ms) Breadth-First Path: [<Node Z>, <Node A>, <Node S>, <Node F>, <Node B>, <Node C>, <Node G>, <Node D>, <Node E>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>, <Node P>, <Node Q>, <Node R>, <Node S>, <Node T>, <Node U>, <Node V>, <Node W>, <Node X>, <Node Y>, <Node Z>]	Generados: 32 Visitados: 21 Costo total: 1284 Tiempo total: 0.5791000723838806 (ms) Depth-First Path: [<Node Z>, <Node A>, <Node T>, <Node L>, <Node M>, <Node D>, <Node C>, <Node B>, <Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node N>, <Node O>, <Node P>, <Node Q>, <Node R>, <Node S>, <Node U>, <Node V>, <Node W>, <Node X>, <Node Y>, <Node Z>]	Generados: 41 Visitados: 35 Costo total: 583 Tiempo total: 1.041000085830688 (ms) Branch and Bound Path: [<Node Z>, <Node A>, <Node S>, <Node R>, <Node P>, <Node Q>, <Node B>, <Node C>, <Node D>, <Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>, <Node U>, <Node V>, <Node W>, <Node X>, <Node Y>, <Node Z>]	Generados: 26 Visitados: 12 Costo total: 583 Tiempo total: 0.35770004987716675 (ms) Branch and Bound with subestimation Path: [<Node Z>, <Node A>, <Node S>, <Node R>, <Node P>, <Node Q>, <Node B>, <Node C>, <Node D>, <Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>, <Node U>, <Node V>, <Node W>, <Node X>, <Node Y>, <Node Z>]

			S>, <Node O>]	<Node R>, <Node S>, <Node O>]	<Node P>, <Node R>, <Node S>, <Node O>]	<Node B>, <Node P>, <Node R>, <Node S>, <Node O>]
3	Giurgiu	Zerind	Generados: 41 Visitados: 34 Costo total: 615 Tiempo total: 0.3970000147819519 (ms) Breadth-First Path: [<Node Z>, <Node A>, <Node S>, <Node F>, <Node B>, <Node G>]	Generados: 32 Visitados: 21 Costo total: 1284 Tiempo total: 0.5791000723838806 (ms) Depth-First Path: [<Node Z>, <Node A>, <Node T>, <Node L>, <Node M>, <Node D>, <Node C>, <Node P>, <Node R>, <Node S>, <Node F>, <Node B>, <Node G>]	Generados: 41 Visitados: 35 Costo total: 583 Tiempo total: 1.0410000085830688 (ms) Branch and Bound Path: [<Node Z>, <Node A>, <Node S>, <Node R>, <Node P>, <Node B>, <Node G>]	Generados: 26 Visitados: 12 Costo total: 583 Tiempo total: 0.35770004987716675 (ms) Branch and Bound with subestimation Path: [<Node Z>, <Node A>, <Node S>, <Node R>, <Node P>, <Node B>, <Node G>]
4	Neamt	Dobreta	Generados: 32 Visitados: 26 Costo total: 765 Tiempo total: 0.27250003814697266 (ms) Breadth-First Path: [<Node D>, <Node C>, <Node P>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]	Generados: 31 Visitados: 19 Costo total: 1151 Tiempo total: 0.35620003938674927 (ms) Depth-First Path: [<Node D>, <Node C>, <Node P>, <Node R>, <Node S>, <Node F>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]	Generados: 32 Visitados: 26 Costo total: 765 Tiempo total: 0.21349996328353882 (ms) Branch and Bound Path: [<Node D>, <Node C>, <Node P>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]	Generados: 23 Visitados: 12 Costo total: 765 Tiempo total: 0.5027000308036804 (ms) Branch and Bound with subestimation Path: [<Node D>, <Node C>, <Node P>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]
5	Mehadi a	Fagaras	Generados: 31 Visitados: 23 Costo total: 520 Tiempo total: 0.27079999446868896 (ms) Breadth-First Path: [<Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>]	Generados: 29 Visitados: 18 Costo total: 928 Tiempo total: 0.18309998512268066 (ms) Depth-First Path: [<Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>]	Generados: 36 Visitados: 27 Costo total: 520 Tiempo total: 0.26190000772476196 (ms) Branch and Bound Path: [<Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>]	Generados: 25 Visitados: 16 Costo total: 520 Tiempo total: 0.3278999924659729 (ms) Branch and Bound with subestimation Path: [<Node E>, <Node F>, <Node G>, <Node H>, <Node I>, <Node J>, <Node K>, <Node L>, <Node M>, <Node N>, <Node O>]

		Costo total: 615 Tiempo total: 0.3970000147819519 (ms) Breadth-First Path: [<Node Z>, <Node A>, <Node S>, <Node F>, <Node B>, <Node G>]	Costo total: 1284 Tiempo total: 0.5791000723838806 (ms) Depth-First Path: [<Node Z>, <Node A>, <Node T>, <Node L>, <Node M>, <Node D>, <Node C>, <Node P>, <Node R>, <Node S>, <Node F>, <Node B>, <Node G>]	Costo total: 583 Tiempo total: 1.0410000085830688 (ms) Branch and Bound Path: [<Node Z>, <Node A>, <Node S>, <Node R>, <Node P>, <Node B>, <Node G>]	Costo total: 583 Tiempo total: 0.35770004987716675 (ms) Branch and Bound with subestimation Path: [<Node Z>, <Node A>, <Node S>, <Node R>, <Node P>, <Node B>, <Node G>]	
4	Neamt	Dobreta	Generados: 32 Visitados: 26 Costo total: 765 Tiempo total: 0.27250003814697266 (ms) Breadth-First Path: [<Node D>, <Node C>, <Node P>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]	Generados: 31 Visitados: 19 Costo total: 1151 Tiempo total: 0.35620003938674927 (ms) Depth-First Path: [<Node D>, <Node C>, <Node P>, <Node R>, <Node S>, <Node F>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]	Generados: 32 Visitados: 26 Costo total: 765 Tiempo total: 0.21349996328353882 (ms) Branch and Bound Path: [<Node D>, <Node C>, <Node P>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]	Generados: 23 Visitados: 12 Costo total: 765 Tiempo total: 0.5027000308036804 (ms) Branch and Bound with subestimation Path: [<Node D>, <Node C>, <Node P>, <Node B>, <Node U>, <Node V>, <Node I>, <Node N>]
5	Mehadi a	Fagaras	Generados: 31 Visitados: 23 Costo total: 520 Tiempo total: 0.27079999446868896 (ms) Breadth-First Path: [<Node F>, <Node S>, <Node R>, <Node C>, <Node D>, <Node M>]	Generados: 29 Visitados: 18 Costo total: 928 Tiempo total: 0.18309998512268066 (ms) Depth-First Path: [<Node F>, <Node B>, <Node P>, <Node R>, <Node S>, <Node A>, <Node T>, <Node L>, <Node M>]	Generados: 36 Visitados: 27 Costo total: 520 Tiempo total: 0.26190000772476196 (ms) Branch and Bound Path: [<Node F>, <Node S>, <Node R>, <Node C>, <Node D>, <Node M>]	Generados: 25 Visitados: 16 Costo total: 520 Tiempo total: 0.3278999924659729 (ms) Branch and Bound with subestimation Path: [<Node F>, <Node S>, <Node R>, <Node C>, <Node D>, <Node M>]



# FIN PRÁCTICA 1

< Universidad de Las Palmas de Gran Canaria >

