Búsqueda no informada con coste variable - El problema del viaje

In [1]:
```
import problema_espacio_estados as probee
import busqueda_espacio_estados as busqee
import copy
```

Hasta ahora hemos trabajado con problemas en los que todas las acciones tienen el mismo coste. Veamos con un ejemplo sencillo, como afecta el coste a los diferentes algoritmos.

Tenemos una serie de ciudades y la distancia entre estas (**no son totalmente fieles a la realidad**). No todas las ciudades están conectadas con todas.

In [2]:
```
ciudades = ['AL', 'CA', 'CO', 'GR', 'HU', 'JA', 'MA', 'SE']
distancias = [[0, 0, 0, 175, 0, 235, 0, 0],
              [0, 0, 0, 0, 0, 0, 240, 125],
              [0, 0, 0, 210, 0, 110, 175, 140],
              [175, 0, 210, 0, 0, 100, 130, 0],
              [0, 0, 0, 0, 0, 0, 0, 95],
              [235, 0, 110, 100, 0, 0, 0, 0],
              [0, 240, 175, 130, 0, 0, 0, 215],
              [0, 125, 140, 0, 95, 0, 215, 0]]
```

Veamos la información que nos proporciona la matriz anterior.

In [3]:
```
for i,c1 in enumerate(ciudades):
    for j,c2 in enumerate(ciudades):
        if distancias[i][j] != 0:
            print('{} es accesible desde {} y están separadas por {} Km.'.format(c1,
```

```
AL es accesible desde GR y están separadas por 175 Km.
AL es accesible desde JA y están separadas por 235 Km.
CA es accesible desde MA y están separadas por 240 Km.
CA es accesible desde SE y están separadas por 125 Km.
CO es accesible desde GR y están separadas por 210 Km.
CO es accesible desde JA y están separadas por 110 Km.
CO es accesible desde MA y están separadas por 175 Km.
CO es accesible desde SE y están separadas por 140 Km.
GR es accesible desde AL y están separadas por 175 Km.
GR es accesible desde CO y están separadas por 210 Km.
GR es accesible desde JA y están separadas por 100 Km.
GR es accesible desde MA y están separadas por 130 Km.
HU es accesible desde SE y están separadas por 95 Km.
JA es accesible desde AL y están separadas por 235 Km.
JA es accesible desde CO y están separadas por 110 Km.
JA es accesible desde GR y están separadas por 100 Km.
MA es accesible desde CA y están separadas por 240 Km.
MA es accesible desde CO y están separadas por 175 Km.
MA es accesible desde GR y están separadas por 130 Km.
MA es accesible desde SE y están separadas por 215 Km.
SE es accesible desde CA y están separadas por 125 Km.
SE es accesible desde CO y están separadas por 140 Km.
SE es accesible desde HU y están separadas por 95 Km.
SE es accesible desde MA y están separadas por 215 Km.
```

Acciónes sin considerar distancia entre ciudades (todas están igual de lejos).

Vamos a hacer una primera versión en la que no consideramos la distancia entre ciudades

In [4]:
```
class Viajar_sin_coste(probee.Accion):
    def __init__(self, desde, hasta, ciudades, distancias):
        super().__init__("Viajar desde {} hasta {}. Distancia: {}".format(ciudades[d
        self.desde = desde
        self.hasta= hasta
        self.ciudades = ciudades
        self.distancia = distancias[desde][hasta]

    def es_aplicable(self, estado):
        return estado == self.desde

    def aplicar(self, estado):
        return self.hasta

#     def coste_de_aplicar(self, estado):
#         return 1
```

In [5]:
```
acciones = [Viajar_sin_coste(i, j, ciudades, distancias)
            for i in range(len(ciudades))
            for j in range(len(ciudades)) if i != j and distancias[i][j] != 0]

estado_inicial = ciudades.index('HU')
estado_final = ciudades.index('AL')
viaje_HU_AL_sin_coste = probee.ProblemaEspacioEstados(acciones, estado_inicial, [est
```

Veamos las soluciones que obtenemos usando diferentes algoritmos de búsqueda no informada.

In [6]:
```
b_profundidad = busqee.BusquedaEnProfundidad(detallado=True)
b_profundidad.buscar(viaje_HU_AL_sin_coste)
```

```
Nodo: Estado: 4; Prof: 0
  Nodo: Estado: 7; Prof: 1
    Nodo: Estado: 6; Prof: 2
      Nodo: Estado: 3; Prof: 3
        Nodo: Estado: 5; Prof: 4
        Nodo: Estado: 0; Prof: 4
```

Out[6]:
```
['Viajar desde HU hasta SE. Distancia: 95',
 'Viajar desde SE hasta MA. Distancia: 215',
 'Viajar desde MA hasta GR. Distancia: 130',
 'Viajar desde GR hasta AL. Distancia: 175']
```

In [7]:
```
b_anchura = busqee.BusquedaEnAnchura(detallado=True)
b_anchura.buscar(viaje_HU_AL_sin_coste)
```

```
Nodo: Estado: 4; Prof: 0
  Nodo: Estado: 7; Prof: 1
    Nodo: Estado: 1; Prof: 2
    Nodo: Estado: 2; Prof: 2
    Nodo: Estado: 6; Prof: 2
      Nodo: Estado: 3; Prof: 3
      Nodo: Estado: 5; Prof: 3
        Nodo: Estado: 0; Prof: 4
```

Out[7]:
```
['Viajar desde HU hasta SE. Distancia: 95',
 'Viajar desde SE hasta CO. Distancia: 140',
 'Viajar desde CO hasta GR. Distancia: 210',
 'Viajar desde GR hasta AL. Distancia: 175']
```

In [8]:
```
b_optima = busqee.BusquedaOptima(detallado=True)
b_optima.buscar(viaje_HU_AL_sin_coste)
```

```
Nodo: Estado: 4; Prof: 0; Valoración: 0; Coste: 0
  Nodo: Estado: 7; Prof: 1; Valoración: 1; Coste: 1
    Nodo: Estado: 1; Prof: 2; Valoración: 2; Coste: 2
    Nodo: Estado: 2; Prof: 2; Valoración: 2; Coste: 2
```

```
    Nodo: Estado: 6; Prof: 2; Valoración: 2; Coste: 2
      Nodo: Estado: 3; Prof: 3; Valoración: 3; Coste: 3
      Nodo: Estado: 5; Prof: 3; Valoración: 3; Coste: 3
        Nodo: Estado: 0; Prof: 4; Valoración: 4; Coste: 4
```

Out[8]:
```
['Viajar desde HU hasta SE. Distancia: 95',
 'Viajar desde SE hasta CO. Distancia: 140',
 'Viajar desde CO hasta GR. Distancia: 210',
 'Viajar desde GR hasta AL. Distancia: 175']
```

Resultados (sin considerar distancias):

- Búsqueda en profundidad:
  - **['HU', 'SE', 'MA', 'GR', 'AL'] 615 Km.**
- Búsqueda en anchura:
  - **['HU', 'SE', 'CO', 'GR', 'AL'] 620 Km.**
- Búsqueda Óptima:
  - **['HU', 'SE', 'CO', 'GR', 'AL'] 620 Km.**

¿Por qué ha obtenido la búsqueda en profundidad mejor resultado?

Acciónes considerarando distancias entre ciudades.

In [9]:
```
class Viajar_con_coste(probee.Accion):
    def __init__(self, desde, hasta, ciudades, distancias):
        super().__init__("Viajar desde {} hasta {}. Distancia: {}".format(ciudades[d
        self.desde = desde
        self.hasta= hasta
        self.ciudades = ciudades
        self.distancia = distancias[desde][hasta]

    def es_aplicable(self, estado):
        return estado == self.desde

    def aplicar(self, estado):
        return self.hasta

    def coste_de_aplicar(self, estado):
        return self.distancia
```

In [10]:
```
acciones = [Viajar_con_coste(i, j, ciudades, distancias)
            for i in range(len(ciudades))
            for j in range(len(ciudades)) if i != j and distancias[i][j] != 0]

estado_inicial = ciudades.index('HU')
estado_final = ciudades.index('AL')
viaje_HU_AL_con_coste = probee.ProblemaEspacioEstados(acciones, estado_inicial, [est
```

Veamos las soluciones que obtenemos usando diferentes algoritmos de búsqueda no informada.

In [11]:
```
b_profundidad = busqee.BusquedaEnProfundidad(detallado=True)
b_profundidad.buscar(viaje_HU_AL_con_coste)
```

```
Nodo: Estado: 4; Prof: 0
  Nodo: Estado: 7; Prof: 1
    Nodo: Estado: 6; Prof: 2
      Nodo: Estado: 3; Prof: 3
        Nodo: Estado: 5; Prof: 4
        Nodo: Estado: 0; Prof: 4
```

Out[11]:
```
['Viajar desde HU hasta SE. Distancia: 95',
 'Viajar desde SE hasta MA. Distancia: 215',
```

```
 'Viajar desde MA hasta GR. Distancia: 130',
 'Viajar desde GR hasta AL. Distancia: 175']
```

In [12]:
```
b_anchura = busqee.BusquedaEnAnchura(detallado=True)
b_anchura.buscar(viaje_HU_AL_con_coste)
```

```
Nodo: Estado: 4; Prof: 0
  Nodo: Estado: 7; Prof: 1
    Nodo: Estado: 1; Prof: 2
    Nodo: Estado: 2; Prof: 2
    Nodo: Estado: 6; Prof: 2
      Nodo: Estado: 3; Prof: 3
      Nodo: Estado: 5; Prof: 3
        Nodo: Estado: 0; Prof: 4
```

Out[12]:
```
['Viajar desde HU hasta SE. Distancia: 95',
 'Viajar desde SE hasta CO. Distancia: 140',
 'Viajar desde CO hasta GR. Distancia: 210',
 'Viajar desde GR hasta AL. Distancia: 175']
```

In [13]:
```
b_optima = busqee.BusquedaOptima(detallado=True)
b_optima.buscar(viaje_HU_AL_con_coste)
```

```
Nodo: Estado: 4; Prof: 0; Valoración: 0; Coste: 0
  Nodo: Estado: 7; Prof: 1; Valoración: 95; Coste: 95
    Nodo: Estado: 1; Prof: 2; Valoración: 220; Coste: 220
    Nodo: Estado: 2; Prof: 2; Valoración: 235; Coste: 235
    Nodo: Estado: 6; Prof: 2; Valoración: 310; Coste: 310
      Nodo: Estado: 5; Prof: 3; Valoración: 345; Coste: 345
      Nodo: Estado: 3; Prof: 3; Valoración: 440; Coste: 440
      Nodo: Estado: 3; Prof: 3; Valoración: 445; Coste: 445
        Nodo: Estado: 0; Prof: 4; Valoración: 580; Coste: 580
```

Out[13]:
```
['Viajar desde HU hasta SE. Distancia: 95',
 'Viajar desde SE hasta CO. Distancia: 140',
 'Viajar desde CO hasta JA. Distancia: 110',
 'Viajar desde JA hasta AL. Distancia: 235']
```

Resultados (considerando distancias):

- Búsqueda en profundidad:
  - **['HU', 'SE', 'MA', 'GR', 'AL'] 615 Km.**
- Búsqueda en anchura:
  - **['HU', 'SE', 'CO', 'GR', 'AL'] 620 Km.**
- Búsqueda Óptima:
  - **['HU', 'SE', 'CO', 'JA', 'AL'] 580 Km.**

Como podemos observar, la búsqueda óptima ha obtenido una solución mejor.