El problema de los misioneros

0.1. EL PROBLEMA DE LOS MISIONEROS --- PRÁCTICA 2 IA GRUPO 5

0.1.1. Definición del problema siguiendo el esquema proporcionado por AIMA

In [1]:
```
# Importamos la librería search de AIMA
from search import *
```

In [2]:
```
# Definición del problema de los misioneros según el esquema de AIMA
class ProblemaMisioneros(Problem):
    ''' Clase problema (formalizacion de nuestro problema) siguiendo la
        estructura que aima espera que tengan los problemas.'''
    def __init__(self, initial, goal=None):
        '''Inicializacion de nuestro problema.'''
        Problem.__init__(self, initial, goal)
        # cada accion tiene un texto para identificar al operador y despues una tupla co
        # cantidad de misioneros y canibales que se mueven en la canoa
        self._actions = [('1c', (0,1)), ('1m', (1, 0)), ('2c', (0, 2)), ('2m', (2, 0)),

    def actions(self, s):
        '''Devuelve las acciones validas para un estado.'''
        # las acciones validas para un estado son aquellas que al aplicarse
        # nos dejan en otro estado valido
        return [a for a in self._actions if self._is_valid(self.result(s, a))]

    def _is_valid(self, s):
        '''Determina si un estado es valido o no.'''
        # un estado es valido si no hay mas canibales que misioneros en ninguna
        # orilla, y si las cantidades estan entre 0 y 3
        return (s[0] >= s[1] or s[0] == 0) and ((3 - s[0]) >= (3 - s[1]) or s[0] == 3) a

    def result(self, s, a):
        '''Devuelve el estado resultante de aplicar una accion a un estado
           determinado.'''
        # el estado resultante tiene la canoa en el lado opuesto, y con las
        # cantidades de misioneros y canibales actualizadas segun la cantidad
        # que viajaron en la canoa
        if s[2] == 0:
            return (s[0] - a[1][0], s[1] - a[1][1], 1)
        else:
            return (s[0] + a[1][0], s[1] + a[1][1], 0)
```

```
In [3]: # Creación del problema dado un estado inicial y un estado objetivo
        estado = ProblemaMisioneros((3, 3, 0), (0, 0, 1))
```

0.1.2. a) Resolved el problema por los distintos métodos de búsqueda vistos:

```
In [4]: # Solución por búsqueda en anchura sin control de estados repetidos
        breadth_first_tree_search(estado).solution()
```

```
Out[4]: [('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2m', (2, 0)),
         ('1m1c', (1, 1)),
         ('2m', (2, 0)),
         ('1c', (0, 1)),
         ('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2c', (0, 2))]
```

```
In [5]: # Solución por búsqueda en anchura con control de estados repetidos
        breadth_first_graph_search(estado).solution()
```

```
Out[5]: [('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2m', (2, 0)),
         ('1m1c', (1, 1)),
         ('2m', (2, 0)),
         ('1c', (0, 1)),
         ('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2c', (0, 2))]
```

```
In [6]: # Solución por búsqueda en profundidad sin control de estados repetidos
        depth_first_graph_search(estado).solution()
```

```
Out[6]: [('1m1c', (1, 1)),
         ('1m', (1, 0)),
         ('2c', (0, 2)),
         ('1c', (0, 1)),
         ('2m', (2, 0)),
         ('1m1c', (1, 1)),
         ('2m', (2, 0)),
         ('1c', (0, 1)),
         ('2c', (0, 2)),
         ('1m', (1, 0)),
         ('1m1c', (1, 1))]
```

```
In [7]: # ------------- NO FINALIZA --------------
        # Solución por búsqueda en profundidad sin control de estados repetidos
        # depth_first_tree_search(estado).solution()
```

0.1.3. b) Analizad el coste de las soluciones encontradas:

Búsqueda en anchura sin control de repetidos: Encuentra una solución que requiere de once operaciones. Además, suponiendo que cuando hablamos del coste nos referimos al número de operaciones ejecutadas (con cada operación por tanto de coste uno), el algoritmo de búsqueda en anchura alcanza siempre una solución óptima. La solución dada requiere el mínimo número de operaciones para llegar al estado objetivo.

Búsqueda en anchura con control de repetidos: Encuentra una solución que requiere de once operaciones, de hecho, encuentra la misma que el algoritmo que no controla nodos repetidos. Esto es lógico porque la naturaleza del algoritmo es la misma, completo, por lo que siempre que exista solución encuentra una solución y óptimo (gracias a que suponemos el coste igual al número de operaciones), por lo que encuentra una solución que requiere el mínimo número de operaciones.

¿Qué diferencia hallamos entonces entre estos dos algoritmos hermanos? El primero al no requerir de estructuras de datos adicionales para el control de nodos repetidos ni requerir de operaciones adicionales de comprobación es un algoritmo válido e incluso más rápido cuando existen soluciones con un bajo número de operaciones. Cuando este no es el caso, el algoritmo analiza árboles de exploración enteros repetidas veces (árboles que crecen exponencialmente) por lo que el coste en tiempo es mayor.

Búsqueda en profundidad sin control de repetidos: En este caso el algoritmo no es capaz de hallar una solución al problema ni de terminar su ejecución. Esto es debido a que al no existir un control de nodos explorados y, más especialmente, a la existencia de operaciones inversas el algoritmo puede caer en bucles infinitos explorando ciclos.

Por ejemplo, la siguiente secuencia no atraviesa estados prohibidos y genera un bucle:
S0 -> Mover dos caníbales derecha -> Mover dos caníbales izquierda -> S0

Búsqueda en profundidad con control de repetidos: Al añadir el control de nodos repetidos el algoritmo nunca examinará un nodo ya explorado lo que impide que se creen bucles infinitos debidos a la exploración de ciclos en el grafo de estados y operaciones de nuestro problema. Con este algoritmo se halla una solución de coste once operaciones por lo que sabemos que es óptima.

0.1.4. c) Analizad el coste en memoria de los distintos algoritmos y la causa de sus diferencias:

```
In [8]: # Ampliamos la clase con atributos para realizar las métricas de memoria:
        class ProblemaConMetricas(Problem):

            """Clase extendida para incorporar atributos para la medida del coste de ejecución""

            def __init__(self, problem):
                self.initial = problem.initial
```

```
            self.problem = problem
            self.analizados  = 0
            self.goal = problem.goal

        def actions(self, estado):
            return self.problem.actions(estado)

        def _is_valid(self, estado):
            return self.problem._is_valid(estado)

        def result(self, estado, accion):
            return self.problem.result(estado, accion)

        def goal_test(self, estado):
            self.analizados += 1
            return self.problem.goal_test(estado)

        def coste_de_aplicar_accion(self, estado, accion):
            return self.problem.coste_de_aplicar_accion(estado,accion)
```

```
In [40]: def resuelve_y_muestra_metricas(problema, algoritmo, h=None):

             if h: sol= algoritmo(problema, h).solution()
             else: sol= algoritmo(problema).solution()

             print("Longitud de la solución: {0}. Nodos analizados: {1}".format(len(sol),problem
```

```
In [41]: problema_misioneros = ProblemaMisioneros((3, 3, 0), (0, 0, 1))
```

Búsqueda en anchura sin control de repetidos:

```
In [11]: %%time
         breadth_first_tree_search(estado).solution()
```

```
CPU times: user 113 ms, sys: 772 µs, total: 114 ms
Wall time: 113 ms
```

```
Out[11]: [('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2m', (2, 0)),
          ('1m1c', (1, 1)),
          ('2m', (2, 0)),
```

```
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2c', (0, 2))]
```

```
In [12]: %%timeit
         breadth_first_tree_search(estado).solution()
```

```
10 loops, best of 3: 67.3 ms per loop
```

```
In [42]: problema_metricas   = ProblemaConMetricas(problema_misioneros)
         resuelve_y_muestra_metricas(problema_metricas, breadth_first_tree_search)
```

```
Longitud de la solución: 11. Nodos analizados: 11878
```

Búsqueda en anchura con control de repetidos:

```
In [50]: %%time
         breadth_first_graph_search(estado).solution()
```

```
CPU times: user 154 µs, sys: 6 µs, total: 160 µs
Wall time: 164 µs
```

```
Out[50]: [('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2m', (2, 0)),
          ('1m1c', (1, 1)),
          ('2m', (2, 0)),
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2c', (0, 2))]
```

```
In [18]: %%timeit
         breadth_first_graph_search(estado).solution()
```

```
10000 loops, best of 3: 79.6 µs per loop
```

```
In [43]: problema_metricas   = ProblemaConMetricas(problema_misioneros)
         resuelve_y_muestra_metricas(problema_metricas, breadth_first_graph_search)
```

```
Longitud de la solución: 11. Nodos analizados: 15
```

Búsqueda en profundidad con control de repetidos:

```
In [51]: %%time
         depth_first_graph_search(estado).solution()

CPU times: user 151 µs, sys: 6 µs, total: 157 µs
Wall time: 160 µs


Out[51]: [('1m1c', (1, 1)),
          ('1m', (1, 0)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2m', (2, 0)),
          ('1m1c', (1, 1)),
          ('2m', (2, 0)),
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1m', (1, 0)),
          ('1m1c', (1, 1))]

In [20]: %%timeit
         depth_first_graph_search(estado).solution()

10000 loops, best of 3: 78.4 µs per loop


In [48]: problema_metricas   = ProblemaConMetricas(problema_misioneros)
         resuelve_y_muestra_metricas(problema_metricas, depth_first_graph_search)

Longitud de la solución: 11. Nodos analizados: 12
```

Búsqueda en profundidad sin control de repetidos:

```
In [ ]: # ------------- NO FINALIZA --------------
        # %%time
        # depth_first_tree_search(estado).solution()

In [ ]: # ------------- NO FINALIZA --------------
        # %%timeit
        # depth_first_tree_search(estado).solution()

In [ ]: # ------------- NO FINALIZA --------------
        # problema_metricas   = ProblemaConMetricas(problema_misioneros)
        # resuelve_y_muestra_metricas(problema_metricas, depth_first_tree_search)
```

0.1.5. d) ¿Qué algoritmo consideras mejor? Razonad la respuesta:

Está claro en esta situación que el algoritmo de búsqueda en profundidad sin control de repetidos no puede ser el mejor por no ser ni siquiera válido. De los restantes observamos que la búsqueda en anchura sin control de repetidos genera también una enorme cantidad de nodos explorados (12000 frente a los 12 de la búsqueda en profundidad con control de repetidos) y que a su vez su tiempo de ejecución es también del orden de mil veces mayor a su versión con control de ciclos. Esto es debido a que gracias a la representación tan sencilla del problema (con a penas tres valores enteros) las comprobaciones son de bajísimo coste algorítmico.

Por tanto, para este problema queda patente que el control de ciclos incrementa el coste de las iteraciones ridículamente y reduce en órdenes exponenciales (en función de la profundidad de las soluciones encontradas) el número de nodos analizados. De los dos alogritmos con control de repeticiones el de búsqueda en profundidad parece ser el mejor debido a la existencia de múltiples soluciones distribuidas por el árbol de exploración. La existencia de soluciones a baja profundidad hace la búsqueda en anchura también una opción válida y eficaz.

0.1.6. Ejercicio opcional. Define alguna heurística y estudia las propiedades del algoritmo A*

La lógica parece indicarnos que cuantos más misioneros y caníbales haya en la orilla derecha más cerca de la solución estamos. Como bien se explica en el temario de la asignatura, dentro de las propias condiciones del problema se especifica que no se puede llevar a más de dos personas en la lancha. Es por esto que para llevar k personas de la orilla izquierda a la derecha necesitaremos como mínimo k/2 viajes en la lancha (de hecho generalmente podríamos crear una heurística más informada gracias a que únicamente tenemos una balsa y, por tanto, los viajes de vuelta serán imprescindibles en ciertos casos). Necesitar como mínimo k/2 viajes (con k el número de personas en la orilla izquierda) nos hace ver que la heurística definida es en efecto consistente.

```
In [29]: ## Heurística elegida: (Número de personas en la orilla izquierda) / (2 = Capacidad de
         def heuristica(node):
             state = node.state
             return (state[0] + state[1])/2
```

```
In [30]: astar_search(problema_misioneros, heuristica).solution()
```

```
Out[30]: [('1m1c', (1, 1)),
          ('1m', (1, 0)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2m', (2, 0)),
          ('1m1c', (1, 1)),
          ('2m', (2, 0)),
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2c', (0, 2))]
```

Estudiamos ahora las propiedades de la heurística dada

```
In [46]: %%time
         astar_search(estado, heuristica).solution()
```

```
CPU times: user 470 µs, sys: 24 µs, total: 494 µs
Wall time: 501 µs
```

```
Out[46]: [('1m1c', (1, 1)),
          ('1m', (1, 0)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2m', (2, 0)),
          ('1m1c', (1, 1)),
          ('2m', (2, 0)),
          ('1c', (0, 1)),
          ('2c', (0, 2)),
          ('1c', (0, 1)),
          ('2c', (0, 2))]
```

```
In [49]: %%timeit
         astar_search(estado, heuristica).solution()
```

```
10000 loops, best of 3: 143 µs per loop
```

```
In [47]: problema_metricas   = ProblemaConMetricas(problema_misioneros)
         resuelve_y_muestra_metricas(problema_metricas, astar_search, heuristica)
```

```
Longitud de la solución: 11. Nodos analizados: 14
```

Los resultados muestran que la búsqueda ciega en profundidad analiza menos nodos que la búsqueda guiada por nuestra heurística. Esto, junto con el mayor coste algorítmico causado por el añadido de complejidad de la heurística hace que también en los tiempos de ejecución se refleje un empeoramiento con respecto a los métodos de búsqueda no informados.

La conclusión que podemos obtener es que, dada la escasa complejidad (en cuanto a tamaño del árbol de exploración) del problema y dada la existencia de soluciones a baja profundidad (con un bajo número de operaciones), el uso de una heurística si bien no supone un empeoramiento sustancial del tiempo ni memoria empleada en la resolución del problema (gracias a que la heurística es de cómputo en coste constante y bajo), si que es innecesaria para este problema en concreto.

Si en vez de tratar el problema de los misioneros (3 misioneros y 3 caníbales) tratásemos el problema de las Jornadas Mundiales de la Juventud (JMJ, con 3000 misioneros y 3000 jóvenes caníbales) el uso de una heurística sería absolutamente imprescindible y los métodos de búsqueda ciega resultarían inaplicables.

Francisco Javier Blázquez Martínez, Boris Carballa Corredoira, Juan Carlos Villanueva Quirós