Modelado del Problema de las Torres de Hanoi

Formas de plantear los estados

Podemos representar el estado s como una tupla de listas que indican dónde se encuentra cada disco:

$$s = (l_1, l_2, l_3), \quad l_i \in \{\text{A}, \text{B}, \text{C}\}$$

Cardinalidad:

$$\begin{aligned} l_1, l_2, l_3 &\Rightarrow \mathbb{Z}^3 \\ d_j \in \{a_1, a_2, \ldots, a_n\}, &\quad \text{con } a_i \in \mathbb{Z} \\ &\Rightarrow \mathbb{Z}^{2^n} \text{ o } \mathbb{Z}^{2^{15}} \text{ para } n = 15 \end{aligned}$$

También podemos usar una tupla ordenada:

$$s = (d_1, d_2, \ldots, d_n), \quad d_i \in \{\text{A}, \text{B}, \text{C}\}$$

Cardinalidad: $|s| = 3^n$
Acciones:

$$A = \{\text{‘AB’}, \text{‘AC’}, \text{‘BA’}, \text{‘BC’}, \text{‘CA’}, \text{‘CB’}\}$$

Usaremos este enfoque.

Clase en Python

Listing 1: Clase TorresHanoi en Python

```
class TorresHanoi(ModeloBusqueda):
    def __init__(self, n):
        self.A = ['AB', 'AC', 'BA', 'BC', 'CA', 'CB']
        self.n = n

    def acciones_legales(self, s):
        if len(s) != self.n or not set(s).issubset({'A', 'B', 'C'}):
            raise ValueError("__")
```

```
        av = []
        for a in self.A:
            de, hacia = a[0], a[1]
            if de in s and (hacia not in s or s.index(de) < s.index(hacia)):
                av.append(a)
        return av

    def transicion(self, s, a):
        costo_local = 1
        if a not in self.acciones_legales(s):
            raise ValueError("__")
        sn = s[:]
        sn[s.index(a[0])] = a[1]
        return sn, costo_local
```

Ejemplos

- $s = [A, A, B, A, C]$

  $\rightarrow$ Acciones legales: `AB, AC, BC`

- $s' = [C, A, B, A, C]$

  $\rightarrow$ Acciones legales: `CA, CB`

- $s'' = [C, C, B, A, C]$