Calcular o Troco

Supõe que tens um número arbitrário de moedas de 50, 20, 10, 5, 2 e 1 cêntimos, e que pretendes dar o troco no valor de N cêntimos, utilizando o menor número de moedas. Formula o problema em Python, seguindo o paradigma do Espaço de Estados e a plataforma aimas-python, de modo a poder resolver o problema de saber quais as moedas a utilizar para formar qualquer troco desejado. Se quiser pode modelar de modo a poder fornecer a lista de moedas com que pode gerar o troco, não ficando obrigado à lista de 50, 20, 10, 5, 2 e 1.

In [1]:
```
from searchPlus import *
```

In [2]:
```
class Troco(Problem):
    """ Vamos ter como estado uma lista de moedas que tem de estar sempre ordenada.
        Notem que podemos ter várias combinações de [10,2,1] mass que todas represent
am
        termos essas 3 moedas e daí a ordenação para que o espaço de estados seja mín
imo.
    """

    def __init__(self, initial=[], goal=87, moedas = [50,20,10,5,2,1] ):
        self.initial, self.goal, self.moedas = initial, goal, moedas

    def actions(self, state):
        """Indicação das peças que vão deslizar..."""
        return [m for m in self.moedas if sum(state)+m <= self.goal]

    def result(self, state, action):
        """adicionar mais uma moeda  `."""
        new = state.copy()
        new.append(action)
        new.sort()
        return new

    def goal_test(self,state):
        return sum(state) == self.goal
```

In [3]:
```
p = Troco(moedas=[10,2,1],goal=23)
```

In [4]:
```
e0=p.initial
print(e0)
```

```
[]
```

In [5]:
```
p.actions(e0)
```

Out[5]:
```
[10, 2, 1]
```

In [6]:
```
e1=p.result(e0,10)
print(e1)
```

```
[10]
```

Testemos se o estado inicial mudou.

In [7]:
```
p.initial
```

Out[7]:
```
[]
```

In [8]:
```
e2=p.result(e1,10)
print(e2)
print(e1)
```

```
[10, 10]
[10]
```

In [9]:
```
p.actions(e2)
```

Out[9]: [2, 1]

In [10]:
```
e3=p.result(e2,1)
print(e3)
```

```
[1, 10, 10]
```

In [11]:
```
p.actions(e3)
```

Out[11]: [2, 1]

In [12]:
```
e4=p.result(e3,2)
print(e4)
```

```
[1, 2, 10, 10]
```

In [13]:
```
p.actions(e4)
```

Out[13]: []

In [14]:
```
p.goal_test(e4)
```

Out[14]: True

Poderíamos formular de uma maneira alternativa que não implicasse uma ordenação das moedas. Usaremos uma tabela (dicionário) que é um contador, as chaves são as moedas e os valores os contadores. Para que possamos somar o valor do troco, faremos um método que pode estar fora da classe a que chamaremos de ***contaTotal()***.

In [15]:
```
def contaTotal(dic):
    soma = 0
    for i in dic.keys():
        soma += i * dic[i]
    return soma
```

In [16]:
```
moedas = {1:2,2:3,5:0,10:0,20:0,50:2}
```

In [17]:
```
contaTotal(moedas)
```

Out[17]: 108

In [18]:
```
class Troco(Problem):
    """ Vamos ter
    """


    def __init__(self, initial={1:0,2:0,5:0,10:0,20:0,50:0}, goal=87, moedas = [50,20
,10,5,2,1]
):
        self.initial, self.goal, self.moedas = initial, goal, moedas

    def actions(self, state):
        """Indicação das peças que vão deslizar..."""
        return [m for m in self.moedas if contaTotal(state)+m <= self.goal]

    def result(self, state, action):
        """adicionar mais uma moeda  `."""
        new = state.copy()
        new[action]+=1
        return new

    def goal_test(self,state):
        return contaTotal(state) == self.goal
```

In [19]:
```
t = Troco()
```

In [20]:
```
print(t.initial)
```

```
{1: 0, 2: 0, 5: 0, 10: 0, 20: 0, 50: 0}
```

In [21]:
```
t.actions(t.initial)
```

Out[21]:
```
[50, 20, 10, 5, 2, 1]
```

In [22]:
```
e1=t.result(t.initial,50)
print(e1)
```

```
{1: 0, 2: 0, 5: 0, 10: 0, 20: 0, 50: 1}
```

In [23]:
```
t.actions(e1)
```

Out[23]:
```
[20, 10, 5, 2, 1]
```

Exercício: Podem fazer uma variante deste exercício que é considerar que não possuem um número infinito de cada uma das moedas mas que elas existem em quantidades limitadas. Por exemplo, calcule o troco de 87 sabendo que têm 1 moeda de 50, 2 moedas de 10, 4 moedas de 2 e 10 moedas de 1 cêntimo.