**The wolf, the cabbage and the goat**

The vcl program finds a way to transport the cabbage, the goat and the wolf to the other side of the river. I used the input (1, 1, 1, ’LEFT’, 0, 0, 0), for the variables tuple(wl ,gl, cl, m, wr, gr, cr), each variables meaning: wl – wolf left, gl – goat left, cl – cabbage left, m – man, wr – wolf right, gr – goat right, cr – cabbage right.

Initially, all are on the LEFT side of the river, inclusive the man. We have to move all on the RIGHT. The result after running the code is:

[(1, 0, 1, 'DREAPTA', 0, 1, 0), # wolf and cabbage on left, goat and man on right, man transported goat to the right side of the river

(1, 0, 1, 'STANGA', 0, 1, 0), # man returns empty on the left side

(0, 0, 1, 'DREAPTA', 1, 1, 0), #man brings wolf on the right side, so the cabbage is on left, and man, wolf and goat on right

(0, 1, 1, 'STANGA', 1, 0, 0), #man returns on left side with the goat

(0, 1, 0, 'DREAPTA', 1, 0, 1), #man takes the cabbage on the right side, it results in goat on left, man, wolf and cabbage on right

(0, 1, 0, 'STANGA', 1, 0, 1), #man returns empty handed on left side, leaving the wolf and the cabbage on right side

(0, 0, 0, 'DREAPTA', 1, 1, 1)] #man takes the goat on the right side, and now all are on the right side.

**Missionaries and Cannibals**

The mc program finds a way to transport all the 3 missionaries and 3 cannibals on the other side of the river, knowing that there must never be more cannibals than missionaries on any side of the river. I used the input (3, 3, ‘STANGA’, 0, 0) for the variables tuple(cl, ml, boat, cr, mr), each one meaning cl – cannibals left, ml – missionaries left, boat – boat, cr – cannibals right, mr – missionaries right.

Initially, all are on the LEFT side of the river, inclusive the man. We have to move all on the RIGHT. The result after running the code is:

[(1, 3, 'DREAPTA', 2, 0), #we move 2 cannibals to the right side

(2, 3, 'STANGA', 1, 0), #we get back to the left side with 1 cannibal

(0, 3, 'DREAPTA', 3, 0), #we take the last 2 cannibals to the right side of the river

(1, 3, 'STANGA', 2, 0), #we turn back with 1 cannibal

(1, 1, 'DREAPTA', 2, 2), #we move 2 missionaries to the right part of the river

(2, 2, 'STANGA', 1, 1), #we turn back with 1 cannibal and 1 missionary

(2, 0, 'DREAPTA', 1, 3), #we move to the right 2 missionaries

(3, 0, 'STANGA', 0, 3), #we move back 1 cannibal to the left side

(1, 0, 'DREAPTA', 2, 3), #take 2 cannibals to the right side

(2, 0, 'STANGA', 1, 3), # turn back with 1 cannibal

(0, 0, 'DREAPTA', 3, 3)] #take the remaining 2 cannibals to the right side of the river

The code used for the second problem is:

from search import Problem

def is\_valid(state):

canibalStanga, misionarStanga, barca, canibalDreapta, misionarDreapta = state

if (canibalStanga > misionarStanga > 0) or (canibalDreapta > misionarDreapta > 0) or\

(canibalStanga < 0) or (canibalDreapta < 0) or (misionarStanga < 0) or (misionarDreapta < 0):

return False

return True

class MC(Problem):

def result(self, state, action):

"""The result of going to a neighbor is just that neighbor."""

return action

def value(self, state):

pass

def \_\_init\_\_(self, initial, goal):

self.goal = goal

self.initial = initial

self.visited\_states = []

Problem.\_\_init\_\_(self, self.initial, self.goal)

def \_\_repr\_\_(self):

return "< State (%s, %s) >" % (self.initial, self.goal)

def goal\_test(self, state):

return state == self.goal

def actions(self, cur\_state):

actions = []

self.visited\_states.append(cur\_state)

if cur\_state[2] == 'STANGA':

# Duce 2 canibali

new\_state = (cur\_state[0] - 2, cur\_state[1], 'DREAPTA', cur\_state[3] + 2, cur\_state[4])

if is\_valid(new\_state):

actions.append(new\_state)

# Duce 1 canibal

new\_state = (cur\_state[0] - 1, cur\_state[1], 'DREAPTA', cur\_state[3] + 1, cur\_state[4])

if is\_valid(new\_state):

actions.append(new\_state)

# Duce 1 canibal si 1 misionar

new\_state = (cur\_state[0] - 1, cur\_state[1] - 1, 'DREAPTA', cur\_state[3] + 1, cur\_state[4] + 1)

if is\_valid(new\_state):

actions.append(new\_state)

# Duce 1 misionar

new\_state = (cur\_state[0], cur\_state[1] - 1, 'DREAPTA', cur\_state[3], cur\_state[4] + 1)

if is\_valid(new\_state):

actions.append(new\_state)

# Duce 2 misionari

new\_state = (cur\_state[0], cur\_state[1] - 2, 'DREAPTA', cur\_state[3], cur\_state[4] + 2)

if is\_valid(new\_state):

actions.append(new\_state)

else:

# Se intoarce cu un canibal

new\_state = (cur\_state[0] + 1, cur\_state[1], 'STANGA', cur\_state[3] - 1, cur\_state[4])

if is\_valid(new\_state):

actions.append(new\_state)

# Se intoarce cu un misionar

new\_state = (cur\_state[0], cur\_state[1] + 1, 'STANGA', cur\_state[3], cur\_state[4] - 1)

if is\_valid(new\_state):

actions.append(new\_state)

# Se intoarce cu 2 misionari

new\_state = (cur\_state[0], cur\_state[1] + 2, 'STANGA', cur\_state[3], cur\_state[4] - 2)

if is\_valid(new\_state):

actions.append(new\_state)

# Se intoarce cu 2 canibali

new\_state = (cur\_state[0] + 2, cur\_state[1], 'STANGA', cur\_state[3] - 2, cur\_state[4])

if is\_valid(new\_state):

actions.append(new\_state)

# Se intoarce 1 canibal si 1 misionar

new\_state = (cur\_state[0] + 1, cur\_state[1] + 1, 'STANGA', cur\_state[3] - 1, cur\_state[4] - 1)

if is\_valid(new\_state):

actions.append(new\_state)

return actions