CP468 – A1

Pillip Lee

Derek Cruise

Scott Cosentino

Aman Kumar

Group 11

2017-10-04

Table of Contents

[Problem Statement 2](#_Toc494837487)

[Problem Formulation 2](#_Toc494837488)

[0. State Notation and Constraints 2](#_Toc494837489)

[1. Initial State 2](#_Toc494837490)

[2. Action / Successor Function 2](#_Toc494837491)

[3. Transition Model 2](#_Toc494837492)

[4. Goal Test 3](#_Toc494837493)

[5. Path Cost 3](#_Toc494837494)

[State Space 4](#_Toc494837495)

[Implementation 5](#_Toc494837497)

[Results 7](#_Toc494837498)

[-Scenario 1- 7](#_Toc494837499)

[-Scenario 2- 8](#_Toc494837500)

[-Scenario 3- 8](#_Toc494837501)

[-Scenario 4- 9](#_Toc494837502)

[-Scenario 5- 10](#_Toc494837503)

# Problem Statement

3 Missionaries and 3 Cannibals are on one side of a river, along with a boat that can hold one or two passengers. Find a way to transport everyone to the other side of the river, without ever leaving a group of Missionaries in one place outnumbered by the Cannibals in that place.

# Problem Formulation

## 0. State Notation and Constraints

Notation: (Number of Cannibals on right bank, Number of missionaries on right bank, boat location)

Constraints:

* Number of cannibals on any side of river bank cannot exceed the number of missionaries on that side.
* Boat cannot travel empty across the river.
* Maximum of two entities can be on boat at any time.

## 1. Initial State

Initial State: (0, 0, Left)

## 2. Action / Successor Function

C = Cannibal, M = Missionary

Function: Carry(Entity 1, Entity 2) -> where Entity is either C, M, or None

Actions:

* Carry(C, None) = 1 Cannibal makes it across the river
* Carry(M, None) = 1 Missionary makes it across the river
* Carry(C, C) = 2 Cannibals make it across the river
* Carry(M, M) = 2 Missionaries make it across the river
* Carry(C, M) = 1 Missionary and 1 Cannibal make it across the river

## 3. Transition Model

Notation: Current State + Carry(Entity 1, Entity 2) = Result

|  |
| --- |
| **Current State + Carry(Entity 1, Entity 2) = Result** |
| (0,0,Left) + Carry(C,None) = (1,0 ,Right) |
| (0,0,Left) + Carry(C,C) = (2,0 ,Right) |
| (0,0,Left) + Carry(C,M) = (1,1,Right) |
| (1,0,Left) + Carry(C,None) = (2,0 ,Right) |
| (1,0,Left) + Carry(C,C) = (3,0 ,Right) |
| (1,0,Left) + Carry(M,None) = (1,1,Right) |
| (1,1,Left) + Carry(C,M) = (2,2,Right) |
| (2,0,Left) + Carry(C,None) = (3,0,Right) |
| (2,0,Left) + Carry(M,M) = (2,2,Right) |
| (2,2,Left) + Carry(M,None) = (2,3,Right) |
| (2,2,Left) + Carry(C,M) = (3,3,Right) |
| (0,3,Left) + Carry(C,None) = (1,3,Right) |
| (0,3,Left) + Carry(C,C) = (2,3,Right) |
| (1,3,Left) + Carry(C,None) = (2,3 ,Right) |
| (1,3,Left) + Carry(C,C) = (3,3,Right) |
| (2,3,Left) + Carry(C,None) = (3,3,Right) |
| (1,0,Right) + Carry(C,None) = (0,0,Left) |
| (1,1,Right) + Carry(M,None) = (1,0,Left) |
| (1,1,Right) + Carry(C,M) = (0,0,Left) |
| (2,0,Right) + Carry(C,None) = (1,0,Left) |
| (2,0,Right) + Carry(C,C) = (0,0,Left) |
| (2,2,Right) + Carry(M,M) = (2,0,Left) |
| (2,2,Right) + Carry(C,M) = (1,1,Left) |
| (3,0,Right) + Carry(C,None) = (2,0,Left) |
| (3,0,Right) + Carry(C,C) = (1,0,Left) |
| (1,3,Right) + Carry(C,None) = (0,3,Left) |
| (2,3,Right) + Carry(C,None) = (1,3,Left) |
| (2,3,Right) + Carry(C,C) = (0,3,Left) |
| (2,3,Right) + Carry(M,None) = (2,2,Left) |
| (3,3,Right) + Carry(C,None) = (2,3,Left) |
| (3,3,Right) + Carry(C,C) = (1,3,Left) |
| (3,3,Right) + Carry(C,M) = (2,2,Left) |

## 4. Goal Test

All cannibals and missionaries on right side of river bank.

Final State (3,3,Right)

## 5. Path Cost

The per trip cost for crossing the river is 1. Therefore, the path cost is the total number of times the river was crossed.

# State Space

# 

# Implementation

MISSIONARIES = 3 #Number of Missionaries at start

CANNIBALS = 3 #Number of Cannibals at start

BOAT\_SIZE = 2 #Maximum number of passengers

#Holds a list of states and ensures no duplicates

class **Data\_Structure**:

states = [] #list of states

current\_id = 0 #current position in the list

#returns the id for a given Cannibal, Missionaries and side combination, -1 if not found

def **Get\_Id**(*self*, c, m, side):

id = -1

for x in range(0, len(*self*.states)):

if *self*.states[x].c == c and *self*.states[x].m == m and *self*.states[x].side == side:

id = x

break

return id

#creates a new state and returns id if it does not already exist, returns the id of the state if it already exists

def **Create\_State**(*self*, c, m, side, depth):

id = *self*.Get\_Id(c, m, side)

if id == -1:

*self*.states.append(State(c, m, side, depth))

id = *self*.current\_id

*self*.current\_id += 1

return id

#Holds the information for each state

class **State**:

c = -1 #Number of Cannibals on the right bank

m = -1 #Number of missionaries on the right bank

side = 0 #Which side the boat is on, 1 for right, -1 for left

depth = -1 #length of shortest path to the state from the initial state

Following\_States = [] #list of the ids of the states that are reachable in one move

def **\_\_init\_\_**(*self*, given\_c, given\_m, given\_side, given\_depth):

*self*.c = given\_c

*self*.m = given\_m

*self*.side = given\_side

*self*.depth = given\_depth

*self*.Following\_States = []

#Adds the given state to the following states list if it is not already there

def **Add\_Follower**(*self*, id):

if id != -1 and id not in *self*.Following\_States:

*self*.Following\_States.append(id)

tree = Data\_Structure()

#Creates a new state if the new state is possible

def **Send\_Boat**(id, c\_over, m\_over, depth):

if c\_over + m\_over > 0 and m\_over + c\_over <= BOAT\_SIZE:

if ((tree.states[id].side-1)\*tree.states[id].c <= (tree.states[id].side-1)\*c\_over and #If sending from the left side, is there enough cannibals

(tree.states[id].side-1)\*tree.states[id].m <= (tree.states[id].side-1)\*m\_over and #If sending from the left side, is there enough missionaries

(tree.states[id].side+1)\*(CANNIBALS-tree.states[id].c) >= (tree.states[id].side+1)\*c\_over and #If sending from the right side, is there enough cannibals

(tree.states[id].side+1)\*(MISSIONARIES-tree.states[id].m) >= (tree.states[id].side+1)\*m\_over and #if sending from the right side, is there enough missionaries

(tree.states[id].m + m\_over\*tree.states[id].side == 0 or #will the missionaries not be outnumbered on the right side

tree.states[id].m + m\_over\*tree.states[id].side >= tree.states[id].c + c\_over\*tree.states[id].side) and

(MISSIONARIES-tree.states[id].m - m\_over \* tree.states[id].side == 0 or #will the missionaries not be outnumbered on the left side

MISSIONARIES-tree.states[id].m - m\_over \* tree.states[id].side >= CANNIBALS-tree.states[id].c - c\_over\*tree.states[id].side)):

temp\_id = tree.Create\_State(tree.states[id].c + c\_over\*tree.states[id].side, tree.states[id].m + m\_over\*tree.states[id].side, tree.states[id].side\*-1, depth)

tree.states[id].Add\_Follower(temp\_id)

optimal\_paths = [] #list of the optimal paths

#populates the optimal\_paths list recursively

def **rec\_find\_optimal**(id, end\_id, path = []):

path.append(id)

if id == end\_id: #if we reached the end state

optimal\_paths.append(path)

elif tree.states[id].depth < tree.states[end\_id].depth: #elif we still are not as deep as the end state

for x in range(0,len(tree.states)): #find a following state which is 1 deeper

if tree.states[x].depth == tree.states[id].depth+1 and x in tree.states[id].Following\_States:

rec\_find\_optimal(x, end\_id, list(path))

#create the initial state

id = tree.Create\_State(0, 0, 1,0)

end\_state\_id = -1

while id < len(tree.states): #loop through all states and try all possible actions to find more states

if tree.states[id].c == CANNIBALS and tree.states[id].m == MISSIONARIES:

end\_state\_id = id #record the id of the end state

for c\_over in range(0,BOAT\_SIZE+1): #try all possible boat combinations

for m\_over in range(0,BOAT\_SIZE-c\_over+1):

Send\_Boat(id, c\_over, m\_over, tree.states[id].depth+1)

id += 1

if end\_state\_id == -1: #if the end state was not found

print(*"Unsolveable"*)

else:

rec\_find\_optimal(0,end\_state\_id) #find all optimal paths

for x in optimal\_paths: #print all optimal paths

path = *""*

pos = 0

while pos+1 < len(x):

path = path + *"C"*\*(tree.states[x[pos+1]].c-tree.states[x[pos]].c) + *"M"*\*(tree.states[x[pos+1]].m-tree.states[x[pos]].m) + *"-"*

pos += 1

if pos+1 < len(x):

path = path + *"C"*\*(tree.states[x[pos]].c-tree.states[x[pos+1]].c) + *"M"*\*(tree.states[x[pos]].m-tree.states[x[pos+1]].m) + *"-"*

pos += 1

print (path[:-1])

# Results

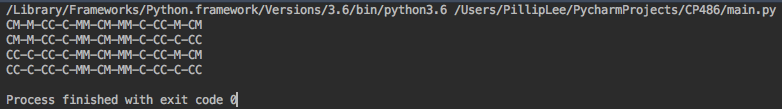
## -Scenario 1-

# of cannibals: 3

# of missionaries: 3

# of boat size: 2

Output:



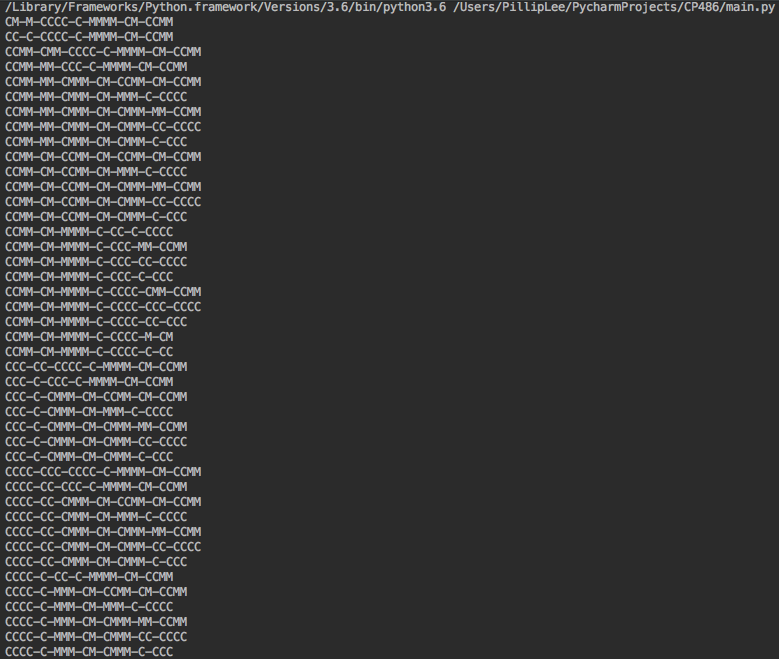
## -Scenario 2-

# of cannibals: 5

# of missionaries: 5

# of boat size: 4

Output:



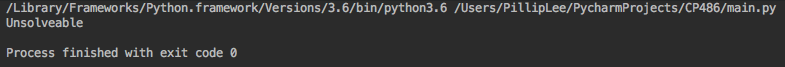
## -Scenario 3-

# of cannibals: 3

# of missionaries: 3

# of boat size: 1

Output:



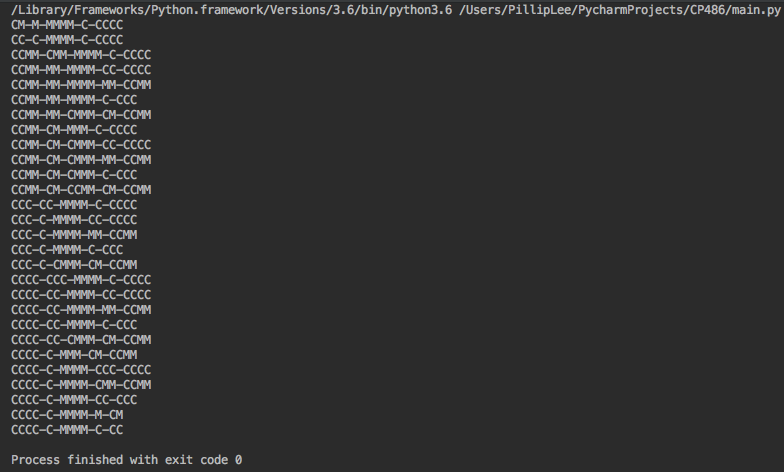
## -Scenario 4-

# of cannibals: 4

# of missionaries: 4

# of boat size: 4

Output:



## -Scenario 5-

# of cannibals: 10

# of missionaries: 10

# of boat size: 9

Output:

