N Missionary And Cannibal

Assignment No. 1

Submitted By:
Ashutosh Chapagain (08)
Krishu Kr. Thapa (58)

Problem Statement

Three missionaries and three cannibals come to the bank of a river they wish to cross. There is a boat that will hold only two and any of the group is able to row. If there are ever more missionaries than cannibals on any side of the river the cannibals will get converted.

Strategies Used To Accomplish the Goal

- Depth-First Search
- Breadth-First Search
- Best-First Search

Assumption:

N=3

All 3 Missionaries and Cannibals are on the left side of the river which indicates boat position 1. Other side represents boat position 0.

Initial State:

3,3,1

Final State:

0,0,0

Constraints:

The no. of cannibals should never exceed the no. of missionaries at any side.

Condition To Satisfy The Constraint:

```
if node.missionary >= 0 and (3-node.missionary) >= 0 \
            and node.cannibal >= 0 and (3-node.cannibal) >= 0 \
            and (node.missionary == 0 or node.missionary >= node.cannibal) \
            and ((3-node.missionary) == 0 or (3-node.missionary) >= (3-node.cannibal)):
                return True
        else:
                return False
```

Depth-First Tree:

Algorithm:
1) Place the node at top of the stack.
2) Pop out the element and find its descendants.
3) Check if the descendant are already explored.
4) Add it to the active list as a stack.
5) If empty, the conclusion is we did not reach goal state.
6) Otherwise goto 2 unless goal state is reached.

Output
In order of
MissionaryLeft,CannibalLeft,BoatPos

```
3,3,1
3,1,0
3,2,1
3,0,0
3,1,1
1,1,0
2,2,1
0,2,0
0,3,1
0,1,0
1,1,1
0,0,0
```

Breadth-First Tree:

Similar to DFS, but here we maintain a queue.

Algorithm:
1) Place the node at top of the queue.
2) Pop out the element and find its descendants.
3) Check if the descendant are already explored.
4) Add it to the active list as a queue.
5) If empty, the conclusion is we did not reach goal state.
6) Otherwise goto 2 unless goal state is reached.

Output
In order of
MissionaryLeft,CannibalLeft,BoatPos

```
3,3,1
2,2,0
3,2,0
3,1,0
3,2,1
3,0,0
3,1,1
1,1,0
2,2,1
0,2,0
0,3,1
```

0,1,0
1,1,1
0,0,0

Best-First Search

A greedy approach is taken to find the most promising descendant of a given node. It can also be said as a greedy approach.
Heuristics Needed : **h(x)= (MissionaryLeft+CannibalLeft)/2**

1) Place the node at top of the list.
2) Pop out the element and find its descendants.
3) Check if the descendant are already explored.
4) Among the unexplored descendants find the most promising one using the heuristic function.
4) Add it to the active list as a list.
5) If empty, the conclusion is we did not reach goal state.
6) Otherwise goto 2 unless goal state is reached.

MissionaryLeft,CannibalLeft,BoatPos
3,3,1
3,1,0
3,2,1
3,0,0
3,2,1
3,1,1
1,1,0
2,2,1
0,2,0
0,3,1
0,1,0
1,1,1
0,0,0

Analysis

Strategies Used	**No. Of Steps Required**
Depth-First Search	11
Breadth-First Search	14
Best- First Search	11

Note: Best-First Tree is still better than Depth-First tree because the steps in DFS depends on the node which appears first. In Best-First Search, it does not depend upon the arrangement of nodes.

Other Variations

If the boat holds 2 people, then 2 couples require 5 trips; with 4 or more couples, the problem has no solution.

Code

```
class DrawGraph:
	def 
__init__(self,myself,child1=None,child2=None,child3=None,child4=None,child5=None,v=None):

		self.myself=myself
		self.child1=child1
		self.child2=child2
		self.child3=child3
		self.child4=child4
		self.child5=child5

	def getAll(self):
		return((self.child1,self.child2,self.child3,self.child4,self.child5))

class Node:
	def __init__(self,missionary,cannibal,boatPos,parent=None):
		self.missionary=missionary
		self.cannibal=cannibal
		self.boatPos=boatPos
		self.parent=parent

	def __eq__(self, other):
		return self.missionary == other.missionary and self.cannibal==other.cannibal and 
self.boatPos==other.boatPos

	def __hash__(self):
		return hash((self.missionary,self.cannibal,self.boatPos))



class Tree:
	def __init__(self,node,num):
		self.node=node
		self.num=num

	def isFinal(self):
		return (self.node.missionary==0)&(self.node.cannibal==0)&(self.node.boatPos==0)


	def getChildrens(self):
		children=[]

		if self.node.boatPos==1:

			node=Node(self.node.missionary-1,self.node.cannibal-
1,1^self.node.boatPos,self.node)

			if checkIfPossible(node,self.num):
				children.append(node)

			node=Node(self.node.missionary,self.node.cannibal-
1,1^self.node.boatPos,self.node)

			if checkIfPossible(node,self.num):
				children.append(node)
```

```
                        node=Node(self.node.missionary-
1,self.node.cannibal,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)

                        node=Node(self.node.missionary-
2,self.node.cannibal,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)

                        node=Node(self.node.missionary,self.node.cannibal-
2,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)


                if self.node.boatPos==0:


        node=Node(self.node.missionary+1,self.node.cannibal+1,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)


        node=Node(self.node.missionary,self.node.cannibal+1,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)


        node=Node(self.node.missionary+1,self.node.cannibal,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)


        node=Node(self.node.missionary+2,self.node.cannibal,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)


        node=Node(self.node.missionary,self.node.cannibal+2,1^self.node.boatPos,self.node)

                        if checkIfPossible(node,self.num):
                                children.append(node)


                return children

        def getEditDistance(self,node):
                return (node.missionary+node.cannibal)/2
```

```
def checkIfPossible(node,x):

        '''
        if node.boatPos==0:
                return (node.missionary>=node.cannibal)&(node.cannibal>=0)&(node.missionary>=0)
        if node.boatPos==1:
                return
(node.missionary<=3)&(node.cannibal<=3)&(node.missionary>=node.cannibal)&(3-
node.missionary<=3-node.cannibal)
        '''

        if node.missionary >= 0 and (x-node.missionary) >= 0 \
            and node.cannibal >= 0 and (x-node.cannibal) >= 0 \
            and (node.missionary == 0 or node.missionary >= node.cannibal) \
            and ((x-node.missionary) == 0 or (x-node.missionary) >= (x-node.cannibal)):
                return True
        else:
                return False

def doBFS(x):

        listOfGraphs=[]

        possibleList=[]

        a=0

        exploredSet=set()

        node=Node(x,x,1,parent=None)

        possibleList.append(node) # adding the first state into the list

        while possibleList:

                a+=1
                currentState=possibleList.pop(0)
                exploredSet.add(currentState)
                tree=Tree(currentState,x)
                if tree.isFinal():
                        return currentState,listOfGraphs
                children=tree.getChildrens()
                '''
                if a==20:
                        for ii in children:
                                print(ii.missionary,ii.cannibal,ii.boatPos)
                        break

                '''
                sons=[]
                for i in children:
                        sons.append(i)
                        if (i not in possibleList) & (i not in exploredSet):
```

```
                        possibleList.append(i)

                len(sons)
                drawGraph=DrawGraph(currentState,*(sons))
                listOfGraphs.append(drawGraph)



        return 0


def doDFS(x):

        listOfGraphs=[]

        possibleList=[]

        a=0

        exploredSet=set()

        node=Node(x,x,1,parent=None)

        possibleList.append(node) # adding the first state into the list

        while possibleList:

                a+=1
                currentState=possibleList.pop()
                exploredSet.add(currentState)
                tree=Tree(currentState,x)
                if tree.isFinal():
                        return currentState,listOfGraphs
                children=tree.getChildrens()
                '''
                if a==20:
                        for ii in children:
                                print(ii.missionary,ii.cannibal,ii.boatPos)
                        break

                '''
                sons=[]
                for i in children:
                        sons.append(i)
                        if (i not in possibleList) & (i not in exploredSet):
                                possibleList.append(i)


                drawGraph=DrawGraph(currentState,*(sons))
                listOfGraphs.append(drawGraph)



        return 0


def doBestFS(num):
    listOfGraphs=[]

    possibleList=[]

    a=0

    exploredSet=set()

    node=Node(num,num,1,parent=None)

    possibleList.append(node) # adding the first state into the list

    while possibleList:
        x={}

        a+=1
        currentState=possibleList.pop(0)
        exploredSet.add(currentState)
        tree=Tree(currentState,num)
        if tree.isFinal():
            return currentState,listOfGraphs
        children=tree.getChildrens()


        sons=[]
        iii=0
        for i in children:
            sons.append(i)
            iii+=1
            if (i not in possibleList) & (i not in exploredSet):
                a=tree.getEditDistance(i)
                x[iii]=a

        XValueMin=min(x.values())

        childrenKey=[key for key in x.keys() if (x[key] == XValueMin)].pop()

        possibleList.append(children[childrenKey-1])



#print(children[x.get(min(x.values()))].missionary,children[x.index(min(x))].cannibal,children[x.index(
min(x))].boatPos)


        drawGraph=DrawGraph(currentState,*(sons))
        listOfGraphs.append(drawGraph)




    return 0
```

```
def bfsManipulations(x):
	state,listOfGraphs=doBFS(x)
	traversedList=[]
	traversedList.append(state)

	parent=state.parent

	while parent:
		traversedList.append(parent)

		parent=parent.parent

	for i in traversedList[::-1]:
		print(i.missionary,i.cannibal,i.boatPos)


	print ("\t \tA Complete BFS Visualization In The Console")
	step=0

	for steps in listOfGraphs:
		step+=1
		print("Step :",step)
		print("      "+"("+str(steps.myself.missionary)+str(steps.myself.cannibal)
+str(steps.myself.boatPos)+")"+"          ")

		a=steps.getAll()

		for i in a:
			try:

				print(" "+"("+str(i.missionary)+str(i.cannibal)+str(i.boatPos)+")"+"
",end='')
			except:
				pass

		print ('\n')
		print ('**********************************************************')

def dfsManipulations(x):

	print('\n\n')
	print('********************************************************')
	state,listOfGraphs=doDFS(x)
	traversedList=[]
	traversedList.append(state)

	parent=state.parent

	while parent:
		traversedList.append(parent)

		parent=parent.parent

	for i in traversedList[::-1]:
		print(i.missionary,i.cannibal,i.boatPos)
	print ("\t \tA Complete DFS Visualization In The Console")
	step=0

	for steps in listOfGraphs:
		step+=1
		print("Step :",step)
		print("     "+"("+str(steps.myself.missionary)+str(steps.myself.cannibal)
+str(steps.myself.boatPos)+")"+"          ")

		a=steps.getAll()

		for i in a:
			try:

				print(" "+"("+str(i.missionary)+str(i.cannibal)+str(i.boatPos)+")"+"
",end='')
			except:
				pass

		print ('\n')
		print ('************************************************')


def bestfsManipulations(x):

	print('\n\n')
	print('************************************************')
	state,listOfGraphs=doBestFS(x)
	traversedList=[]
	traversedList.append(state)

	parent=state.parent

	while parent:
		traversedList.append(parent)

		parent=parent.parent

	for i in traversedList[::-1]:
		print(i.missionary,i.cannibal,i.boatPos)


	print ("\t \tA Complete Best First Visualization In The Console")
	step=0

	for steps in listOfGraphs:
		step+=1
		print("Step :",step)
		print("     "+"("+str(steps.myself.missionary)+str(steps.myself.cannibal)
+str(steps.myself.boatPos)+")"+"          ")

		a=steps.getAll()

		for i in a:
			try:
```

```
                        print(" "+"("+str(i.missionary)+str(i.cannibal)+str(i.boatPos)+")"+"
",end='')
                  except:
                        pass

            print ('\n')
            print ('***************************************************')




def main():

      x=int(input("Enter the no. of missionaries and cannibals "))
      bfsManipulations(x)
      dfsManipulations(x)

      bestfsManipulations(x)



main()
```