**Aim: Write a program to implement a Water Jug Problem using Python and to solve a Water Jug Problem by using BFS (without using any libraries or packages of python).**

* **Program should be written in generalized way to solve by using any capacity of jug.**
* **Find the minimum number of steps to reach any the below-mentioned goal states.**
* **Find execution time of BFS algorithm. (Only “time” and “random” packages can be used in python if it is necessary to use)**

**Program:**

import time

import random

class node:

  def \_\_init\_\_(self,parentNode):

    self.jug1=0

    self.jug2=0

    self.parentNode=parentNode

def operation(i,inputNode,visitedNodeList):

  x=inputNode.jug1

  y=inputNode.jug2

  result=x+y

  if(i==1 and x<jug1):

    x=jug1

  elif(i==2 and y<jug2):

    y=jug2

  elif(i==3 and x>0):

    x=0

  elif(i==4 and y>0):

    y=0

  elif(i==5 and 0 < result >= jug1 and y>0):

    y=(y-(jug1-x))

    x=jug1

  elif(i==6 and 0 < result >= jug2 and x>0):

    x=(x-(jug2-y))

    y=jug2

  elif(i==7 and 0 < result <= jug1 and y >= 0):

    x=result

    y=0

  elif(i==8 and 0 < result <= jug2 and x >= 0):

    y=result

    x=0

  if(x==inputNode.jug1 and y==inputNode.jug2):

    return None

  if([x,y] not in visitedNodeList):

    newNode = node(inputNode)

    newNode.jug1=x

    newNode.jug2=y

    return newNode

  return None

def generateNode(nodeData, method, visitedNodeList):

  nodeList=[]

  rng = range(1,9)

  if(method.upper()=="DFS"):

    rng = random.sample(range(1,9), 8)

  for i in rng:

    genNode = operation(i,nodeData,visitedNodeList)

    if(genNode!=None):

      nodeList.append(genNode)

  return nodeList

class blindSearch:

  def \_\_init\_\_(self):

    self.nodeList=[]

  def find(self,initNode,destinationNode,method):

    self.nodeList.append(initNode)

    visitedNodeList=[]

    if(method.upper()=="DFS"):

      while len(self.nodeList)!=0:

        tempNode = self.nodeList.pop()

        visitedNodeList.append([tempNode.jug1,tempNode.jug2])

        if(tempNode.jug1 == destinationNode.jug1 and tempNode.jug2 == destinationNode.jug2):

          return [tempNode,len(visitedNodeList),len(visitedNodeList)+len(self.nodeList)]

        else:

          self.nodeList.extend(generateNode(tempNode, "DFS",visitedNodeList))

    elif(method.upper()=="BFS"):

      while len(self.nodeList)!=0:

        tempNode = self.nodeList.pop(0)

        visitedNodeList.append([tempNode.jug1,tempNode.jug2])

        if(tempNode.jug1 == destinationNode.jug1 and tempNode.jug2 == destinationNode.jug2):

          return [tempNode,len(visitedNodeList),len(visitedNodeList)+len(self.nodeList)]

        else:

          self.nodeList.extend(generateNode(tempNode, "BFS",visitedNodeList))

    else:

      print("The search method name is incorrect.")

    return [None,len(visitedNodeList),len(visitedNodeList)+len(self.nodeList)]

def printPath(getNode):

  nodeList=[getNode]

  tempNode = getNode.parentNode

  while tempNode!=None:

    nodeList.append(tempNode)

    tempNode=tempNode.parentNode

  return [reversed(nodeList),len(nodeList)-1]

posibleNode=False

try:

  jug1 = int(input("\n\nEnter the volume of Jug1: "))

  if(not jug1>=0):

    raise Exception("Sorry, no numbers below zero")

  jug2 = int(input("Enter the volume of Jug2: "))

  if(not jug2 >= 0):

    raise Exception("Sorry, no numbers below zero")

  initJug1 = int(input("Enter the initial value of Jug1: "))

  if(not initJug1>=0):

    raise Exception("Sorry, no numbers below zero")

  else:

    if(not initJug1<=jug1):

      raise Exception("Invalid initial value of Jug1")

  initJug2 = int(input("Enter the initial value of Jug2: "))

  if(not initJug2 >= 0):

    raise Exception("Sorry, no numbers below zero")

  else:

    if(not initJug2 <= jug2):

      raise Exception("Invalid initial value of Jug2")

  initNode = node(None)

  initNode.jug1 = initJug1

  initNode.jug2 = initJug2

  destinationNode = node(None)

  print("Select Goal Jug")

  print("1. Jug1")

  print("2. Jug2")

  print("3. Both")

  selectJug=int(input("Select: "))

  if(selectJug==1):

    goalJug1 = int(input("Enter the Jug1 goal value: "))

    if(goalJug1<=jug1):

      posibleNode=True

      destinationNode.jug1=goalJug1

    else:

      raise Exception("Goal Jug1 value not possible. It must be less than or equle to volume")

  elif(selectJug==2):

    goalJug2 = int(input("Enter the Jug2 goal value: "))

    if(goalJug2<=jug2):

      posibleNode=True

      destinationNode.jug2=goalJug2

    else:

      raise Exception("Goal Jug2 value not possible. It must be less than or equle to volume")

  elif(selectJug==3):

    goalJug1 = int(input("Enter the Jug1 goal value: "))

    if(goalJug1<=jug1):

      posibleNode=True

      destinationNode.jug1=goalJug1

    else:

      raise Exception("Goal Jug1 value not possible. It must be less than or equle to volume")

    goalJug2 = int(input("Enter the Jug2 goal value: "))

    if(goalJug2<=jug2):

      posibleNode=True

      destinationNode.jug2=goalJug2

    else:

      raise Exception("Goal Jug2 value not possible. It must be less than or equle to volume")

  else:

    raise Exception("Error: Invalid selection; please try again!")

  if(posibleNode):

    print("\n\n====== BFS is Run ======\n")

    initTime = time.time()

    result = blindSearch().find(initNode,destinationNode,"BFS")

    finishTime = time.time()

    if(result[0]!=None):

      print("Solution is....")

      pathList = printPath(result[0])

      for i in pathList[0]:

        print(str.format("( {0} , {1} )",i.jug1,i.jug2))

      print(str.format("Path Cost: {0}",pathList[1]))

    else:

      print("The solution is not possible!")

    print(str.format("Number of node visited: {0}",result[1]))

    print(str.format("Number of node created: {0}",result[2]))

    print(str.format("Time required for BFS: {:.3f} ms\n",(finishTime-initTime)\*1000))

    print("====== DFS is Run ======\n")

    initTime = time.time()

    result = blindSearch().find(initNode,destinationNode,"DFS")

    finishTime = time.time()

    if(result[0]!=None):

      print("Solution is....")

      pathList = printPath(result[0])

      for i in pathList[0]:

        print(str.format("( {0} , {1} )",i.jug1,i.jug2))

      print(str.format("Path Cost: {0}",pathList[1]))

    else:

      print("The solution is not possible!")

    print(str.format("Number of node visited: {0}",result[1]))

    print(str.format("Number of node created: {0}",result[2]))

    print(str.format("Time required for DFS: {:.3f} ms\n\n",(finishTime-initTime)\*1000))

except ValueError:

  print("Invalid Value: Only an integer value is allowed.")

except Exception as ex:

  print(ex)

**Output:**

