Water Jug Problem using BFS

Abhirup Chakravarty 17BCE7055

# CODE:

"""

Water jug Problem solution using BFS on state space search.

-Abhirup Chakravarty, 17BCE7055

"""

import copy

class found(Exception):

  pass

class jug(object):

  def \_\_init\_\_(self, cap):

    self.cap = cap

    self.amt = 0

  def fill\_it(self):

    self.amt = self.cap

  def empty(self):

    self.amt = 0

  def transfer(self, jug):

    remainder = jug.fill(self.amt)

    self.amt = remainder

  def fill(self, amt):

    self.amt += amt

    if self.amt > self.cap:

      remainder = self.amt - self.cap

      self.amt -= remainder

      return remainder

    else:

      return 0

  def \_\_repr\_\_(self):

    return "jug: %i/%i" % (self.amt, self.cap)

class jState(object):

  def \_\_init\_\_(self, jugs):

    self.jugs = [copy.copy(j) for j in jugs]

    self.next\_states = []

    self.parent = None

    self.gflag = False

  def add\_state(self, jugs\_state):

    self.next\_states.append(jugs\_state)

    jugs\_state.parent = self

  def is\_same(self, state):

    for i, j in enumerate(self.jugs):

      if j.cap == state.jugs[i].cap and j.amt != state.jugs[i].amt:

        return False

    return True

  def is\_full(self, amt):

    for j in self.jugs:

      if j.amt == amt:

        return True

    return False

class j\_graph(object):

  def \_\_init\_\_(self, jugs, goal):

    self.start = jState(jugs)

    self.goal = goal

    self.graphed = False

  def state\_exists(self, state):

    def fn1(self\_state):

      these\_states = [state.is\_same(s) for s in self\_state.next\_states]

      child\_states = [fn1(s) for s in self\_state.next\_states]

      exists = False

      for ts in these\_states:

        exists = exists or ts

      for cs in child\_states:

        exists = exists or cs

      return exists

    this\_state = state.is\_same(self.start)

    child\_states = fn1(self.start)

    return this\_state or child\_states

  def build\_graph(self):

    def fn1(curr\_st):

      try:

        for i,j in enumerate(curr\_st.jugs):

          fill\_state = jState(curr\_st.jugs)

          fill\_state.jugs[i].fill\_it()

          if not self.state\_exists(fill\_state):

            curr\_st.add\_state(fill\_state)

          if fill\_state.is\_full(self.goal):

            fill\_state.gflag = True

            raise found()

          empty\_state = jState(curr\_st.jugs)

          empty\_state.jugs[i].empty()

          if not self.state\_exists(empty\_state):

            curr\_st.add\_state(empty\_state)

          if empty\_state.is\_full(self.goal):

            empty\_state.gflag = True

            raise found()

          for k in range(len(curr\_st.jugs)):

            if i == k:

              continue

            intermittent\_st = jState(curr\_st.jugs)

            intermittent\_st.jugs[i].transfer(intermittent\_st.jugs[k])

            if not self.state\_exists(intermittent\_st):

              curr\_st.add\_state(intermittent\_st)

            if intermittent\_st.is\_full(self.goal):

              intermittent\_st.gflag = True

              raise found()

        for s in curr\_st.next\_states:

          if s.gflag:

            raise found()

          fn1(s)

      except found:

        pass

    self.graphed = True

    fn1(self.start)

  def print\_solutions(self):

    self.solution\_number = 0

    self.solutions = []

    def trav\_st(curr\_st):

      if curr\_st.gflag:

        self.solution\_number += 1

        print("Solution %i" % self.solution\_number)

        state\_list = get\_path(curr\_st)

        for s in state\_list:

          print(s.jugs)

        print("%i steps" % len(state\_list))

        self.solutions.append({

          "steps": len(state\_list),

          "list": state\_list

        })

      else:

        for s in curr\_st.next\_states:

          trav\_st(s)

    def get\_path(curr\_st, curr\_path = []):

      curr\_path = [curr\_st] + curr\_path

      if curr\_st.parent:

        return get\_path(curr\_st.parent, curr\_path)

      else:

        return curr\_path

    if self.graphed:

      trav\_st(self.start)

      if self.solution\_number == 0:

        print("No solution found.")

      else:

        print("\nBest solution found:")

        min\_steps = 9999999999

        solution = None

        for s in self.solutions:

          if s["steps"] < min\_steps:

            min\_steps = s["steps"]

            solution = s

        for s in solution["list"]:

          print(s.jugs)

        print("%i steps" % solution["steps"])

    else:

      print("Didn't build graph.")

  def \_\_repr\_\_(self):

    def fn1(states, prefix = "-"):

      string = ""

      for s in states:

        string += prefix + str(s.jugs)

        if s.gflag:

          string += " <----"

        string += "\n"

        string += fn1(s.next\_states, "-" + prefix)

      return string

    string = str(self.start.jugs)

    if self.start.gflag:

      string += " <----"

    string += "\n"

    string += fn1(self.start.next\_states)

    return string

def main():

  jugs = input("Input each jug size, separated by space, then press enter.\n")

  goal = input("Input target amt: ")

  jugs = [jug(int(x)) for x in jugs.split()]

  goal = int(goal)

  graph = j\_graph(jugs, goal)

  graph.build\_graph()

  print(graph)

  graph.print\_solutions()

if \_\_name\_\_ == '\_\_main\_\_':

  main()

# OUTPUT:

Input each jug size, separated by space, then press enter.

3 4

Input target amt: 2

[jug: 0/3, jug: 0/4]

-[jug: 3/3, jug: 0/4]

--[jug: 0/3, jug: 3/4]

---[jug: 3/3, jug: 3/4]

----[jug: 2/3, jug: 4/4] <----

--[jug: 3/3, jug: 4/4]

-[jug: 0/3, jug: 4/4]

--[jug: 3/3, jug: 1/4]

---[jug: 0/3, jug: 1/4]

----[jug: 1/3, jug: 0/4]

-----[jug: 1/3, jug: 4/4]

------[jug: 3/3, jug: 2/4] <----

Solution 1

[jug: 0/3, jug: 0/4]

[jug: 3/3, jug: 0/4]

[jug: 0/3, jug: 3/4]

[jug: 3/3, jug: 3/4]

[jug: 2/3, jug: 4/4]

5 steps

Solution 2

[jug: 0/3, jug: 0/4]

[jug: 0/3, jug: 4/4]

[jug: 3/3, jug: 1/4]

[jug: 0/3, jug: 1/4]

[jug: 1/3, jug: 0/4]

[jug: 1/3, jug: 4/4]

[jug: 3/3, jug: 2/4]

7 steps

Best solution found:

[jug: 0/3, jug: 0/4]

[jug: 3/3, jug: 0/4]

[jug: 0/3, jug: 3/4]

[jug: 3/3, jug: 3/4]

[jug: 2/3, jug: 4/4]

5 steps