1 Overview

In Python, write a program that takes three integers:

- the size (volume) of Jar1
- the size (volume) of Jar2
- the desired quantity (volume) of fluid (must be less than Jar2)

That prints out the steps to get the desired quantity (volume) of fluid. For example, if you have two jars that measure 3 liters and 5 liters respectively (Jar1 & Jar2), how would you measure exactly 4 liters of fluid using only those Jars?

INPUT
Size (volume) of Jar1: **3**
Size (volume) of Jar2: **5**
Desired quantity (volume) of fluid: **4**

Table 1: Input data example

OUTPUT	
Jar1: **3** Jar2: **0**	**START:** fill Jar1 with 3 liters
Jar1: **0** Jar2: **3**	pour 3 liters from Jar1 to Jar2
Jar1: **3** Jar2: **3**	refill Jar1 with 3 liters
Jar1: **1** Jar2: **5**	pour 2 liters from Jar1 into Jar2; fill Jar2
Jar1: **1** Jar2: **0**	empty Jar2
Jar1: **0** Jar2: **1**	pour the remaining 1 liter in Jar1 into Jar2
Jar1: **3** Jar2: **1**	refill Jar1 with 3 liters
Jar1: **0** Jar2: **4**	**END:** pour 3 liters from Jar1 into Jar2

Table 2: Output data

Please find working example on GitHub.

1.1 Code

```
import networkx as nx

def func( jars, jars_size):

        if( jars[0] < jars_size[0] ):
                yield jars_size[0], jars[1]

        # Fill jug2
        if( jars[1] < jars_size[1]):
                yield jars[0], jars_size[1]

        # Fill jug1 to jug2
        toFill = min(jars[0], jars_size[1] - jars[1])
        if( toFill > 0 ):
                yield jars[0] - toFill, jars[1] + toFill

        # Pour jug2 to jug1
        toFill = min(jars[1], jars_size[0] - jars[0] )
        if( toFill > 0 ):
                yield jars[0] + toFill, jars[1] - toFill
```

```
        #empty jug1
        if( jars[0] > 0 ):
                yield 0, jars[1]

        if( jars[1] > 0 ):
                yield jars[0], 0

Graph = nx.DiGraph()

def add_connection(Graph, jar, newjar, jar_size):
    if not Graph.has_edge(jar, newjar):
        Graph.add_edge(jar, newjar)
        build_gallon_graph(Graph, newjar, jar_size)

def build_gallon_graph(Graph, jar, jar_size):
    for newjar in func(jar, jar_size):
        add_connection(Graph, jar, newjar, jar_size)

sizes = raw_input("Please enter the Jar1 & Jar2 capacities (separated by a ' '): ").split(' ')
goal = int(raw_input("Enter the desired quantity (volume) of fluid: "))

sizes = ( int(sizes[0]) , int(sizes[1]) )


start = (0,0)
build_gallon_graph( Graph, start , sizes )
path = nx.shortest_path( Graph, start , (0,goal) )

for i in range(0, max(sizes)+1):
        if( i <= sizes[0]):
                if( not Graph.has_node((i,goal)) ):
                        continue
                newPath = nx.shortest_path(Graph, start, (i,goal) )
                if( len( newPath ) < len(path) ):
                        path = newPath
        if( i <= sizes[1]):
                if( not Graph.has_node( (goal,i) ) ):
                        continue
                newPath = nx.shortest_path(Graph, start, (goal,i) )
                if( len( newPath ) < len(path) ):
                        path = newPath

for item in path:
        print ("Jug1: {} and Jug2: {}".format(item[0], item[1]))
```