**Unit 3 Algorithmics**

**Week 1 – Submit Task**

Part 1: Complete the coding task below, on Trinket at <https://trinket.io/python3/45ac0d9744>. (Alternatively copy and paste into VS Code)

import random

# This code creates two random numbers between 1 and 50

first = random.randint(1,50)

second = random.randint(1,50)

# ADD YOUR CODE in the relevant spaces below.

print(*f*"My random numbers are: {first} and {second}")

# 2. Store the numbers in variables 'larger' and 'smaller'. State the larger number.

# (if they are the same then larger = smaller.)

smaller = min(first, second)

larger = max(first, second)

# 3. State which of your numbers are even.

even = print("1") if (first % 2 == 0) else print("2") if (second % 2 == 0) else print("Both") if (first % 2 == 0) and (second % 2 == 0) else print("None")

# 4. State whether or not the larger number is exactly divisible by the smaller

#    number. If not, state the remainder.

div = print("True") if larger % smaller else print(larger % smaller)

# 5. If the two sides are the hypotenuse and shorter side of a right angled triangle,

#    find the length of the other shorter side.

short\_side = print(larger\*\*2 - smaller\*\*2)

# Run your code multiple times to make sure it works as expected.

Part 2: Water Jug Puzzle

Consider the following measuring liquid puzzle. Suppose you have a 3 litre jug and a 5 litre jug, and access to a tap with unlimited water. The jugs do not have measurement marks on them.

1. How would you measure out exactly 1 litre of water?

Solution Steps:

o Fill Jug 2 -> (0, 5)

o Pour Jug 2 into Jug 1 -> (3, 2)

o Empty Jug 2 -> (3, 0)

o Pour Jug 1 into Jug 2 -> (0, 3)

o Fill Jug 1 -> (3, 3)

o Pour Jug 1 into Jug 2 -> (1, 5)

1. How do you express the solution for someone else to understand and follow?

Psuedocode could be used, or a network map could be drawn to visually represent the solution

1. Is your solution the most efficient? That is can it be done in fewer steps?

Probably not, BFS may be more efficient in this case as it will explore all surface level paths before moving a section deeper hence ensuring that the solution uses the shortest path

1. What other amounts of water can you measure in this way?

Any, the code can measure out any valid amount of water

1. What happens if we change the 5 litre jug to a 6 litre jug?

State Debug: (0, 0), Actions: []

State Debug: (0, 6), Actions: ['Fill Jug 2 -> (0, 6)']

State Debug: (3, 3), Actions: ['Fill Jug 2 -> (0, 6)', 'Pour Jug 2 into Jug 1 -> (3, 3)']

State Debug: (3, 0), Actions: ['Fill Jug 2 -> (0, 6)', 'Pour Jug 2 into Jug 1 -> (3, 3)', 'Empty Jug 2 -> (3, 0)']

State Debug: (0, 3), Actions: ['Fill Jug 2 -> (0, 6)', 'Pour Jug 2 into Jug 1 -> (3, 3)', 'Empty Jug 2 -> (3, 0)', 'Pour Jug 1 into Jug 2 -> (0, 3)']

State Debug: (3, 6), Actions: ['Fill Jug 2 -> (0, 6)', 'Pour Jug 2 into Jug 1 -> (3, 3)', 'Empty Jug 2 -> (3, 0)', 'Fill Jug 2 -> (3, 6)']

No solution found.

"""Part 2: Water Jug Puzzle

Consider the following measuring liquid puzzle. Suppose you have a 3 litre jug and a 5 litre jug, and access to a tap with unlimited water. The jugs do not have measurement marks on them.

a. How would you measure out exactly 1 litre of water?

b. How do you express the solution for someone else to understand and follow?

c. Is your solution the most efficient?  That is can it be done in fewer steps?

d. What other amounts of water can you measure in this way?

e. What happens if we change the 5 litre jug to a 6 litre jug?

"""

*class* WaterJugPuzzle:

*def* \_\_init\_\_(*self*, *j1\_cap*, *j2\_cap*, *target*, *show\_dfs\_tree*=False):

*self*.j1\_cap = *j1\_cap*

*self*.j2\_cap = *j2\_cap*

*self*.target = *target*

*self*.visited = set()

*self*.stack = []

*self*.show\_dfs\_tree = *show\_dfs\_tree*

*def* fill\_jug(*self*, *state*, *jug*):

        j1, j2 = *state*

        if *jug* == 1:

            return (*self*.j1\_cap, j2)

        elif *jug* == 2:

            return (j1, *self*.j2\_cap)

*def* empty\_jug(*self*, *state*, *jug*):

        j1, j2 = *state*

        if *jug* == 1:

            return (0, j2)

        elif *jug* == 2:

            return (j1, 0)

*def* pour\_water(*self*, *state*, *from\_jug*, *to\_jug*):

        j1, j2 = *state*

        amount\_to\_pour = min(j1, *self*.j2\_cap - j2) if *from\_jug* == 1 else min(j2, *self*.j1\_cap - j1)

        if *from\_jug* == 1:

            return (j1 - amount\_to\_pour, j2 + amount\_to\_pour)

        elif *from\_jug* == 2:

            return (j1 + amount\_to\_pour, j2 - amount\_to\_pour)

*def* is\_valid\_state(*self*, *state*):

        return 0 <= *state*[0] <= *self*.j1\_cap and 0 <= *state*[1] <= *self*.j2\_cap #ensure jugs are between 0 and capacity

*def* print\_dfs\_tree(*self*, *state*, *actions*):

        if *self*.show\_dfs\_tree:

            print(*f*"State Debug: {*state*}, Actions: {*actions*}")

*def* solve\_puzzle(*self*):

        initial\_state = (0, 0)

*self*.stack.append((initial\_state, []))

        while *self*.stack:

            cstate, actions = *self*.stack.pop()

            if cstate not in *self*.visited:

*self*.visited.add(cstate)

*self*.print\_dfs\_tree(cstate, actions)

                if cstate[0] == *self*.target or cstate[1] == *self*.target:

                    return actions

                all\_actions = [

                    (*self*.fill\_jug(cstate, 1), actions + [*f*'Fill Jug 1 -> {*self*.fill\_jug(cstate, 1)}']),

                    (*self*.fill\_jug(cstate, 2), actions + [*f*'Fill Jug 2 -> {*self*.fill\_jug(cstate, 2)}']),

                    (*self*.empty\_jug(cstate, 1), actions + [*f*'Empty Jug 1 -> {*self*.empty\_jug(cstate, 1)}']),

                    (*self*.empty\_jug(cstate, 2), actions + [*f*'Empty Jug 2 -> {*self*.empty\_jug(cstate, 2)}']),

                    (*self*.pour\_water(cstate, 1, 2), actions + [*f*'Pour Jug 1 into Jug 2 -> {*self*.pour\_water(cstate, 1, 2)}']),

                    (*self*.pour\_water(cstate, 2, 1), actions + [*f*'Pour Jug 2 into Jug 1 -> {*self*.pour\_water(cstate, 2, 1)}'])

                ]

                for nxtstate, nxtaction in all\_actions:

                    if *self*.is\_valid\_state(nxtstate):

*self*.stack.append((nxtstate, nxtaction))

        return None

#Drivercode

jug\_puzzle = WaterJugPuzzle(*j1\_cap*=3, *j2\_cap*=5, *target*=1, *show\_dfs\_tree*=True)

solution\_actions = jug\_puzzle.solve\_puzzle()

if solution\_actions:

    print("\nSolution Steps:")

    for action in solution\_actions:

        print("o " + action)

else:

    print("No solution found.")