HW6 - CS430 Jinyang Li

1. Let's consider 3 Jugs: Ja, Jb, Jc

They all have different capacities, which Ja < Jb <Jc and Jc is filled full of water, Ja and Jb is empty.

Each time we do the operation (search) with three Jugs, there are only 6 cases:

Pour Jc's water to Ja; Pour Jc's water to Jb
Pour Jb's water to Ja; Pour Jb's water to Jc
Pour Ja's water to Jb; Pour Ja's water to Jc

Before perform the next pouring the water between Jugs, the test should perform to check if we got the target specific water in at least one Jug also the status of each Jug under loop:

Foreach loop i :
    Foreach loop j:
        if Ji is empty, it cannot give any water to other one (Jj);
        if Jj is full, it cannot accept any water from other one(Ji);
        if Jj remains water but not full, there are another test should perform: compare the available space of Jj with water in Ji (< or >).

The linked object is needed to store the status of each Jug during each water moving (path in graph). We use linked list here to store the vertex we got. We add all vertex not accorded before and test them one by one so it's same way of using as queue in BFS.

At the same time, after each loop we would check if the target is achieved. After reach target, the list object is used for printing out previous paths (status). And this path is the shortest path.

2.

If the algorithm in P1 is NOT correct, then the path is NOT the shortest path. (the steps we got target is not the least steps.)

Let's consider:

the target status of Jugs is in triple (i,j,k) with shortest path (shorter than algorithm in P1)

1) the triple is the same as we got in P1's algorithm. It would never been enqueued to queue.
2) the triple is like (j,i,k), the loop would end early since we got target before. Because j i k happens before i j k, the i j k would never put in linked list.

So the algorithm is correct.

3.
The time complexity is O(c1 c2 c3)

4.
See program source file for testing instructions. (waterJug.py)

5.

a)

Proof (⇒):
Assume that the graph has an Eulerian path. This means every vertex that has an edge adjacent to it (i.e., every non-isolated vertex) must lie on the tour, and is therefore connected with all other vertices on the tour. This proves that the graph is connected (except for isolated vertices). Next note that, for each vertex in the tour except the first (and last), the walk leaves it just after entering. Thus, every time the tour visits a vertex, it traverses exactly two edges (an even number). Since the Eulerian tour uses each edge exactly once, this implies that every vertex except the first has even degree. And the same is true of the first vertex v as well because the first edge leaving v can be paired up with the last edge returning to v. This completes the proof of this direction.

Proof (⇐):
If every vertex has even degree, we would always have ways go in and go out.
If we have 2 odd vertex, it must done be beginning on be adding.
Because each Eulerian path can be combined with many small Eulerian path,so if there only have one odd node means every other nodes is even and can go through as an small Eulerian path.
But if we move this small Eulerian path, it should be left another Eulerian path,but actually there only exists one node with one line.So it's impossible.

it is impossible that if only one vertex has odd degree.

b)

```
Euler-Circuit (initial_v) {

      foreach near_vertex {

            if (edge(initial, v) never been marked)
                  markEdge(initial, v)
                  initial_degree + 1
                  Euler-Circuit(v)
                  push edge into stack
            end if

      }

}
```