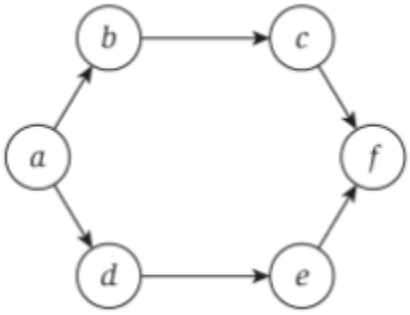
Problem 1: Topological Ordering

Topological ordering is when all nodes in a directed graph can be written in such a way that, going from left to right, all connections are from left to right.

For example, if you have the graph



That graph can be written as:

a, b, c, d, e, f
a, d, e, b, c, f
a, b, d, c, e, f
a, b, d, e, c, f
a, d, b, c, e, f
a, d, b, e, c, f

These are valid topological orderings, because b and d are always after a. c and e are always after b and d respectively, and f is always after c and e.

Topological orderings, if they exist, can be found by performing a DFS on every node in the graph, such that the algorithm will look through all possible neighbors until no more exist at which point the algorithm is finished. In this example we would do

```
DFS(A)
        DFS(B)
                DFS(C)
                        DFS(F)
        DFS(D}
                DFS(E)
```

The topological ordering is the reverse finishing order of each DFS. This means you will need a way of keeping track of when each DFS call finishes using some kind of data structure. Printing these values in reverse order will give the topological ordering.

Note: The DFS algorithm described here is different than the one written in class. In class, the algorithm took in a start and an end. This algorithm will only take in a start and keep looking for neighbors until there's nowhere left to look.

Problem 2: Brick Walls

You have been provided with a class that represents one layer of a brick wall in the form of a list of integers. These integers represent the length of each brick in the first layer of the wall. We want to figure out if we can create a second layer given bricks of a certain size. The constraint being that no two brick endings can line up with each other. Meaning that if you have a wall with two bricks 3 and 1. You can't add 2, 1, and 1 to the second layer because then the 3 and the 1 would end in the same place (note the lengths have to be the same, so the last bricks can (and have to) end in the same place).

You should create a Configuration such that it can find a valid second layer, if one exists. Your Configuration should have a BrickWall and a List<Interger> of valid brick sizes. Both these things should be passed into the constructor (see the tests).

Note: The BrickWall class can determine if a brick placement is valid. Brickwall also contains a copy constructor.