Template - Recursion

There are two ways to implement DFS. The first one is to do recursion which you might already be familiar with. Here we provide a template as reference:

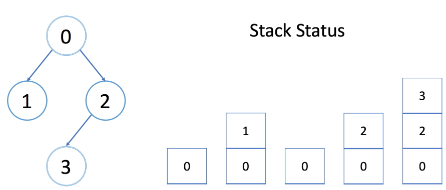
Java Copy

```
/*
 * Return true if there is a path from cur to target.
 */
boolean DFS(Node cur, Node target, Set<Node> visited) {
    return true if cur is target;
    for (next : each neighbor of cur) {
        if (next is not in visited) {
            add next to visted;
            return true if DFS(next, target, visited) == true;
        }
    }
    return false;
}
```

It seems like we don't have to use any stacks when we implement DFS recursively. But actually, we are using the implicit stack provided by the system, also known as the Call Stack (https://en.wikipedia.org/wiki/Call_stack).

An Example

Let's take a look at an example. We want to find a path between node 0 and node 3 in the graph below. We also show you the stack's status during each call.



In each stack element, there is an integer `cur`, an integer `target`, a reference to array `visited` and a reference to array `edges`, which are exactly the parameters we have in the DFS function. We only show `cur` in the stack above.

Each element costs constant space. And the size of the stack is exactly the depth of DFS. So in the worst case, it costs O(h) to maintain the system stack, where h is the maximum depth of DFS. You should never forget to take the system stack into consideration when calculating the space complexity.

> In the template above, we stop when we find the `first` path.
>
> What if you want to find the `shortest` path?
>
> Hint: Add one more parameter to indicate the shortest path you have already found.