How to find dead ends efficiently

A node is a dead end if it has no out-link, or all of its out-links point to dead ends only. To find dead end efficiently (in $O(n+m)$ time), you should use two arrays (of lists):

- Array N^+, where $N^+[i]$ is a *list* of nodes that *links to* i. In other words, $N^+[i]$ is the set of in-neighbors of i. Note that $len(N^+[i])$ is the number of in-links of i.
- Array N^-, where $N^-[i]$ is a *list* of nodes that i *links to*. In other words, $N^-[i]$ is the set of out-neighbors of i. Note that $len(N^-[i])$ is the number of out-links of i.

In addition, you should use an array D to store the out-degree of i. That is, $D[i] = leng(N^-[i])$. Dead ends would have $D[i] = 0$. Also, you should use a Queue, say q (a FIFO queue) to store temporary dead-ends during the execution of the algorithm below:

```
FindDeadEnds(G(V,E), N+, N-)
    initialize array D so that D[i] = 0 for all node i
    initialize an empty Queue q.
    for each node i ∈ V
        D[i] = len(N-[i])
        if D[i] = 0
            put i to q      // i is a dead end
    initialize an empty list L
    while q is not empty
        i ← pop an element from q
        if i is not in L
            append i to the end of L
            for each j in N+[i]
                D[j] ← D[j] − 1      // remove i will decrease out-deg of j
                if D[j] = 0      // j is a new dead end
                    put j to Queue q
    return L
```

The return list L would be the set of dead ends. Their order in L is their removal order.

Tip: to check whether a node i in L, you just need to use a boolean array, say M. Initially, every node i has $M[i] =$ `False`. Every time you put a node i to L, mark $M[i] =$ `True`.