Detecting whether a tree-like data structure contains a cycle

devblogs.microsoft.com/oldnewthing/20230906-00

September 6, 2023

Raymond Chen

Suppose you are given a tree-like data structure, in the sense that you are given a root node, and each node has children. You want to validate that the data structure indeed represents a tree, with no cycles.

You can actually solve this problem by combining two things you already know how to do.

First, you can walk the tree and generate elements. If the necessary node-traversal methods are available, You can use one of the iterative tree-walking algorithms. Otherwise, you can use a traditional depth-first search with a stack. This converts the tree-like structure into a linear list.

Next, you can use Floyd's two-pointer algorithm for finding a cycle in a linked list. This will tell you whether your tree loops back on itself.

Bingo, problem solved by combining two things you already know.

Bonus chatter: If you have to manage an explicit stack, then the memory usage can reach twice the number of nodes in the worst case. Alternatively, you can build a hash table of nodes that have been visited so far, and stop if you ever revisit a node before reaching the end of the tree. This caps the memory usage to the number of nodes.

Bonus bonus chatter: If you know the total number *N* of nodes, say, say because it's recorded in the file format, or you can count them while you load the file, then you don't need to use Floyd's two-pointer cycle-finding algorithm. You can just walk through the tree until you either (1) reach the end, or (2) have walked through *N* nodes. Because once you reach node *N* + 1, you know that you must have revisited a node due to the pigeonhole principle, and therefore found a cycle.