Princeton University

COS 217: Introduction to Programming Systems

DT Algorithms

1. int DT_insert(const char* pcPath);

Step 1). If the DT is not initialized, return INITIALIZATION_ERROR.

Step 2) Build a Path object for the path that we want to insert. Find farthest Node reachable from the root following the given path (static function DT_traversePath).

> If traversal fails, free the path and return the failure status back up to the caller.
>
> If traversal result (Node_curr) is NULL, but the root is not NULL,
> then return CONFLICTING_PATH.
>
> Calculate the "depth" of Node_curr. (Implicitly 0 if Node_curr is NULL)
>
> If Node_curr is the node with the path we want, return ALREADY_IN_TREE.
>
> Otherwise, Node_curr is going to be Parent of the next node to be added

Step 3) Starting at Node's depth + 1 (or at 1 if Node is NULL), for each depth until we reach the depth of the final path that we want to insert:

> Create a new Path object that is a prefix at that depth of the final path
>
> Create a new node at that depth (Node_new) with the new Path as its path and Node_curr as its parent. If Node_new fails, free the paths, free any nodes we've already made here in step 3, and return the failure status back up to the caller.
>
> Otherwise, the new node is now Node_curr. Keep track of the new nodes we've added here in step 3 (in case one eventually fails and we have to delete them all)
>
> Continue with the next iteration of Step 3 at the next depth.

Step 4) Once we have added all the new nodes to reach our final path that we wanted:

> if the root is NULL, set the first new node we made to be the root.
>
> Add the number of new nodes to the data structure's count state variable
>
> Return SUCCESS.

2. int DT_rm(const char *pcPath);

Step 1). Get a pointer to the Node with the path we want to remove. (Use helper function DT_findNode, which traverses path as far as it can, then returns SUCCESS and sets a pointer to the Node found in the traversal only if "as far is it can" is "all the way"). Otherwise, return the error status returned by findNode:

If the DT is not initialized, findNode returns INITIALIZATION_ERROR.

If findNode can't create a Path object with the path, it returns the error status.

If findNode doesn't get all the way to path we want, it returns NO_SUCH_PATH

Step 2). Call Node_free on the Node found by findNode in order to delete the entire hierarchy rooted at that node. Node_free will return the number of nodes removed

Step 3). Node_free removes its parameter Node from that Node's parent's list of children. This will disconnect the Node from the DT.

If the parameter Node has no parent (i.e., it's the root), this step is not done.

Step 4) For every element in Node's list of children, call Node_free recursively on that Child (i.e., goto step 2, but on a child instead of the Node returned by findNode) to remove that sub-hierarchy.

Accumulate the return values from all recursive calls to count total number of nodes removed.

Step 5) Once all Node's children have been recursively destroyed:

free the now-empty DynArray

free Node's path object

free the Node itself

Return the total count of nodes removed (including this Node itself).

Step 6) Once all the recursion finishes and the Node_free call on the original pointer found by findNode returns (finally!) back to DT_rm:
decrement count of the DT by the number of nodes removed
If the count is now 0 (i.e., we removed the root), set the DT's root to NULL
Return SUCCESS