IMPLEMENTATION SPECS

Team: {Dict}ators

Santhosh Reddy Mylaram (2018101030)
Vinay Kumar Tadepalli (2018101048)

Problem Statement:

Build an In-memory key-value pair in c++ without using STL / Boost / DS libraries.

Functionalities :

1. Bool get(key,&val) -

Searches key in the key-value pairs and assigns value to the val variable.Returns false if the key doesn't exist

2. Bool put(key,val) -

Creates a new key and assigns the value val to it. Returns true if the val is overwritten.

3. Bool del(key) -

Deletes the corresponding key-value pair.

4. Bool get(N,&key,&val) -

Searches the Nth key-value pair in Lexicographical order(according to strcmp() function in c++) and assigns them to the pointers Key and val respectively.

5. Bool del(N) -

Deletes the Nth key-value pair in Lexicographical order

Data Structures to be used :

Tries :

The trie stores all the keys in lexicographical order. Each node in the trie has 52 children and the height of the trie would be 64. Each path from the root to a node represents a string i.e; a key and the node stores the corresponding value of the key.

<u>Each node stores the following information :</u>

1. char - Stores a character of the key.
2. Value - Stores the corresponding value of the key.
3. Leaf count - Maintains the count of no of leaves in the subtree with the current node as root.
4. Node *child[52] - Contains the pointers to the children.

Each node represents a key obtained by the concatenation of all the characters of it's Ancestors.

Implementing Functionalities:

put(key,val) :

Traverses through the trie to put the value to the corresponding key. If the key already exists it replaces the previous value and returns true. If the key doesn't exist, It creates a new node with the corresponding value to the key.

The complexity of this functionality is O(h) where h is the height of the tree. Because in the worst case, we have to traverse from the root to the leaf.

get (key, &val) :

Traverses through the trie to find the key and assigns the corresponding value to val and returns true if the key exists. Otherwise returns false.

The complexity of this functionality is O(h) where h is the height of the tree. Because in the worst case, we have to traverse from the root to the leaf.

del(key):

Traverses through the trie to find the key and deletes the corresponding node in the trie.

The complexity of this functionality is O(h) where h is the height of the tree. Because in the worst case, we have to traverse from the root to the leaf.

get(N,&key,&val) :

Traverses through the trie to find the Nth smallest key in Lexicographical order and assigns the corresponding key and value to the pointers key and val.

The normal time complexity of this would be O(n), where n is the total no. of nodes in the trie, Because in the worst case we have to iterate over all the elements to find the Nth smallest string. But this complexity can be reduced as discussed in the net section.

del(N) :

Traverses through the trie to find the Nth smallest key in Lexicographical order and deletes the corresponding key-value pair ,i.e; the corresponding node in the trie.

The normal time complexity of this would be O(n), where n is the total no. of nodes in the trie, Because in the worst case we have to iterate over all the elements to find the Nth smallest string. But this complexity can be reduced as discussed in the net section.

Optimizing Lexicographic searches:

As we are maintaining the leaf count for each node, we can directly skip some subtrees and directly search the subtree in which the Nth smallest key is located. So, the complexity of these operations can be reduced from O(n) to O(h), where n is the total no of keys and h is the height of the tree.

Why we chose TRIE:

The Complexity of all functionalities are O(h) where h is the height of the tree and maximum value of h is 64 as maximum length of key is 64.So,each is performed in constant time.

Memory is also not wasted,we just create what we need i.e When the put() call happens.

We didn’t go for hashing because to search Nth key value complexity raises upto O(n) where n is total no.of keys present.And also to perform put(),get() and del() in constant time,all keys possible have to be hashed in the upfront which causes wastage of memory as we create memory for which we may not use in future.