Patricia Markel Trie API Documentation

Version 1.0.1

Created By

Rahul Shelke

This document is for internal purposes only and subjects to updates over a period of time as the system improves.

Node Types

The Trie consists of following types of nodes:

1. EmptyNode: Null Node / Empty String
2. BranchNode: Node with 17 items to traverse
3. LeafNode: Node with key and value. Key here is encoded remaining path
4. ExtensionNode: Node with shared keys and next node reference.

Object Type

To call any of the APIs, create an object of VTrie(), class.

```
VTrie trie = new VTrie();
trie.insert('a711355', 5409.00)
```

APIs for Trie:

1. insert(key, value)
2. select(key)
3. select(rootHash, key)
4. update(key, value)
5. delete(key)
6. getRoot()

Insert / Add

Insert acts like an insert query from traditional RDS with the only difference being it accepts only the key-value pair to be stored in the trie.

Call:

insert(key, value)

Parameters:

key: Key to be stored in the trie

Value: Transaction value.

Returns:

True if key-value pair stored successfully, else will return False

E.g:

```
result = trie.insert('a711355', 5409.00)
if(result) {
      std::cout << "Successfully stored key-value pair." <<
std::endl;
} else {
      std::cout << "Failed to store key-value pair." << std::endl;
}
```

Select / Get

Select API acts as a select query from traditional RDS with the only difference being it accepts only the key to be searched in the trie.

Call:

select(key)

Parameters:

key: Key to be searched in the trie

Returns:

If the matching key is found, it will return the value stored at the Leaf Node.

E.g:

```
result = trie.select('a711355')
std::cout << result << std::endl;
```

NOTE: The above API will start searching from the rootHash always.

Select

This is similar to the above select, the only difference is it allows us to search from any node from which we would like to search for key n trie.

Call:

select(rootHash, key)

Parameters:

rootHash: Hash of the node from which to start searching for the key.

key: Key to be searched in the trie

Returns:

If the matching key is found, it will return the value stored at the Leaf Node.

E.g:

Let's say current node under processing is having has 'a7' as it's shared nibble, with the memory address of 100203, to search for key '11355' the call will look as follows:

```
result = trie.select('a7', '11355')
std::cout << result << std::endl;
```

Update

Updates the value present at `key` in the trie.

Call:

update(key, value)

Parameters:

key: Key to search in the trie

Value: Transaction value.

Returns:

True if value at specified key is updated successfully, else will return False

E.g:

```
result = trie.update('a711355', 5337.54)
if(result) {
      std::cout << "Successfully updated value of the key." <<
std::endl;
} else {
      std::cout << "Failed to update the value." << std::endl;
}
```

Delete

Deletes the node specified by key, iff it is a leaf node or empty node. The delete operation gives an error if the key specified indicates a node as a Branch Node or an Extension Node.

Call:

delete(key)

Parameters:

key: Key to search in the trie

Returns:

True if value at specified key node is a leaf node or an empty node and is deleted successfully, else will return False

E.g:

```
result = trie.delete('a711355')
if(result) {
      std::cout << "Transaction is deleted." << std::endl;
} else {
      std::cout << "Failed to delete the transaction." <<
std::endl;
}
```

Get Root

Gives the root hash of the trie.

Call:

getRoot()

Parameters:

Returns:

Root Hash value of the trie, if trie exists else NULL.

E.g:

```
result = trie.getRoot()
if(result) {
      std::cout << root << std::endl;
} else {
      std::cout << “Failed to get the root, either trie is empty
or some error occured during processing.” << std::endl;
}
```