1. Aiken Trie

graph TD

ROOT((Root)) -->|c| C(C)

C -->|a| A(A)

A -->|r| R(R)

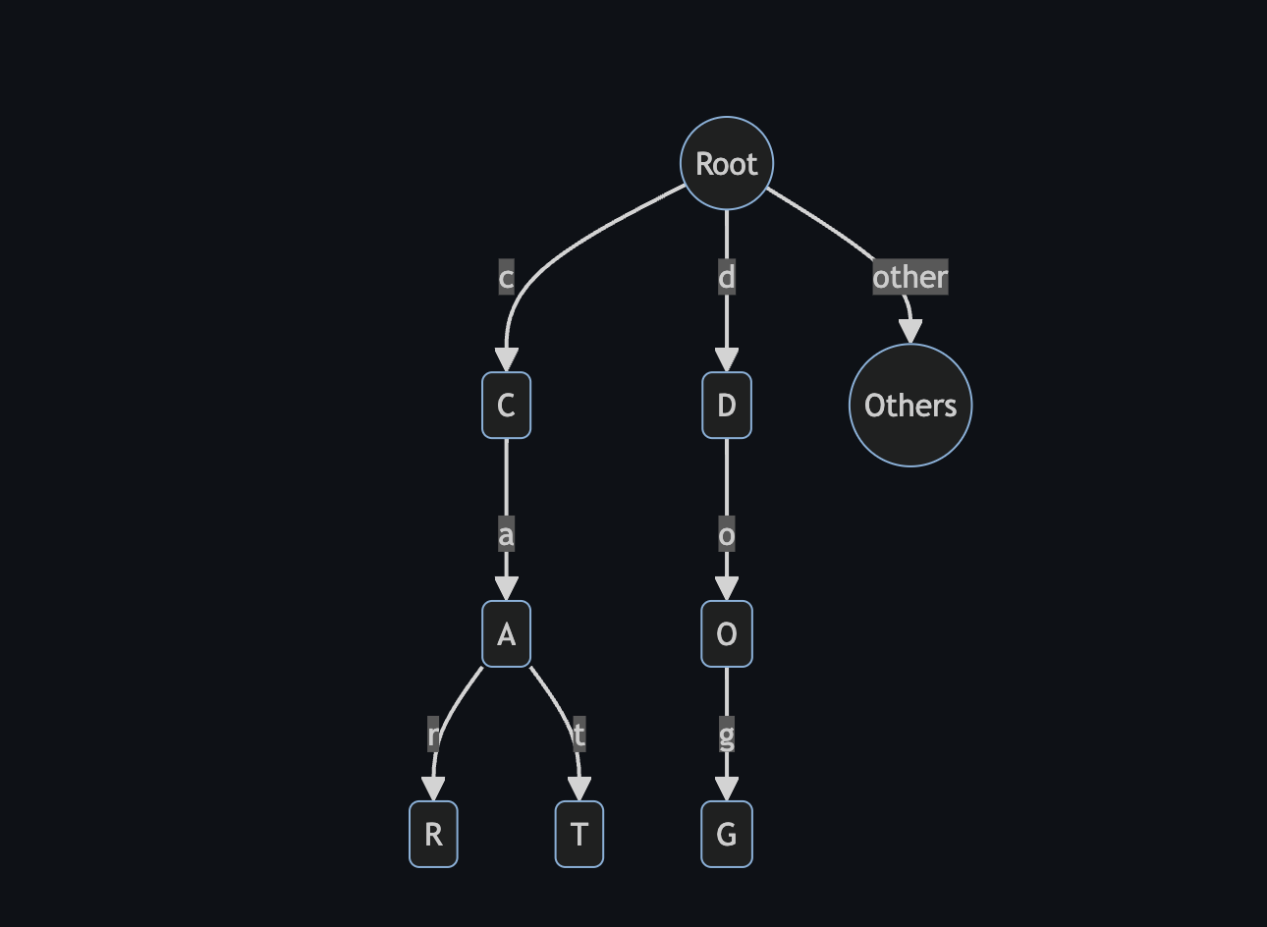
A -->|t| T(T)

ROOT -->|d| D(D)

D -->|o| O(O)

O -->|g| G(G)

ROOT -->|other| OTH((Others))



Test Result:

import { Emulator, Lucid } from "lucid-cardano";

import { defaultProtocolParams } from "./constants";

import { generateAccount } from "./utils";

import { trieScript } from "../src/const";

import {

appendTrie,

betweenTrie,

createTrie,

getTrieOrigin,

getUtxoByKey,

} from "../src";

console.log(`Trie Script Size: ${trieScript.script.length / 2}`);

describe("Synthetics", () => {

let emulator: Emulator;

let lucid: Lucid;

let trieAddress: string;

let trieRewardAddress: string;

beforeEach(async () => {

emulator = new Emulator([]);

lucid = await Lucid.new(emulator);

trieAddress = lucid.utils.validatorToAddress(trieScript);

trieRewardAddress = lucid.utils.validatorToRewardAddress(trieScript);

const TRIE\_USER = await generateAccount({

lovelace: BigInt(100000000000000000),

});

emulator = new Emulator([TRIE\_USER], defaultProtocolParams);

emulator.chain[trieRewardAddress] = {

registeredStake: true,

delegation: { rewards: BigInt(0), poolId: "" },

};

lucid = await Lucid.new(emulator);

lucid.selectWalletFromPrivateKey(TRIE\_USER.privateKey);

});

it("Should pass e2e on creation, two insertions", async () => {

let trie = await createTrie(lucid, trieAddress, trieRewardAddress);

emulator.awaitBlock(1);

let trieOrigin = await getTrieOrigin(lucid, trie.trieUnit, trieAddress);

let trieUtxo = await getUtxoByKey(lucid, trie.trieUnit, "", trieAddress);

await appendTrie(

lucid,

trie.trieUnit,

trieOrigin!,

trieUtxo!,

"hello\_world",

trieAddress,

trieRewardAddress,

);

emulator.awaitBlock(1);

let newTrieUtxo = await getUtxoByKey(lucid, trie.trieUnit, "", trieAddress);

await betweenTrie(

lucid,

trie.trieUnit,

trieOrigin!,

newTrieUtxo!,

"hello",

trieAddress,

trieRewardAddress,

);

});

it("Should fail on duplicate insertion (1)", async () => {

let trie = await createTrie(lucid, trieAddress, trieRewardAddress);

emulator.awaitBlock(1);

let trieOrigin = await getTrieOrigin(lucid, trie.trieUnit, trieAddress);

let trieUtxo = await getUtxoByKey(lucid, trie.trieUnit, "", trieAddress);

await appendTrie(

lucid,

trie.trieUnit,

trieOrigin!,

trieUtxo!,

"hello\_world",

trieAddress,

trieRewardAddress,

);

emulator.awaitBlock(1);

let newTrieUtxo = await getUtxoByKey(lucid, trie.trieUnit, "", trieAddress);

betweenTrie(

lucid,

trie.trieUnit,

trieOrigin!,

newTrieUtxo!,

"hello\_world",

trieAddress,

trieRewardAddress,

)

.then(() => {

throw "This should have failed";

})

.catch(() => {

// failure successful

});

});

it("Should fail on duplicate insertion (2)", async () => {

let trie = await createTrie(lucid, trieAddress, trieRewardAddress);

emulator.awaitBlock(1);

let trieOrigin = await getTrieOrigin(lucid, trie.trieUnit, trieAddress);

let trieUtxo = await getUtxoByKey(lucid, trie.trieUnit, "", trieAddress);

await appendTrie(

lucid,

trie.trieUnit,

trieOrigin!,

trieUtxo!,

"hello\_world",

trieAddress,

trieRewardAddress,

);

emulator.awaitBlock(1);

let newTrieUtxo = await getUtxoByKey(lucid, trie.trieUnit, "", trieAddress);

betweenTrie(

lucid,

trie.trieUnit,

trieOrigin!,

newTrieUtxo!,

"hello\_world",

trieAddress,

trieRewardAddress,

)

.then(() => {

throw "This should have failed";

})

.catch(() => {

// failure successful

});

});

});