Assignment 1: Ethereum Data Structures & Proof-of-Inclusion

- Manav Chaudhari CUID: C15795269

A. Design Choices

implemented a simplified Ethereum-style Modified Merkle Patricia Trie (MPT) to store and verify transactions. The design focused on the following three key aspects:

- **Serialization (RLP encoding):**
  Recursive Length Prefix (RLP) is used to turn data into a fixed format. This makes everything compact and consistent. A small helper like rlp_encode handles bytes, numbers, and lists by adding the right length prefixes.

- **Nodes (Leaf, Extension, Branch):**
  - Leaf nodes store the final value (a transaction) along with the remaining path.
  - Extension nodes compress long shared prefixes and point to a child node.
  - Branch nodes have 16 slots (for hex nibbles) and can optionally store a value.
    Together, these node types allow the trie to handle keys efficiently and avoid duplication.

- **Trie construction:**
  The TransactionTrie class organizes all transactions. Keys (transaction indices) are converted to nibble paths, and values (transactions encoded in RLP) are inserted. The trie root acts as a fingerprint for the entire set of transactions.

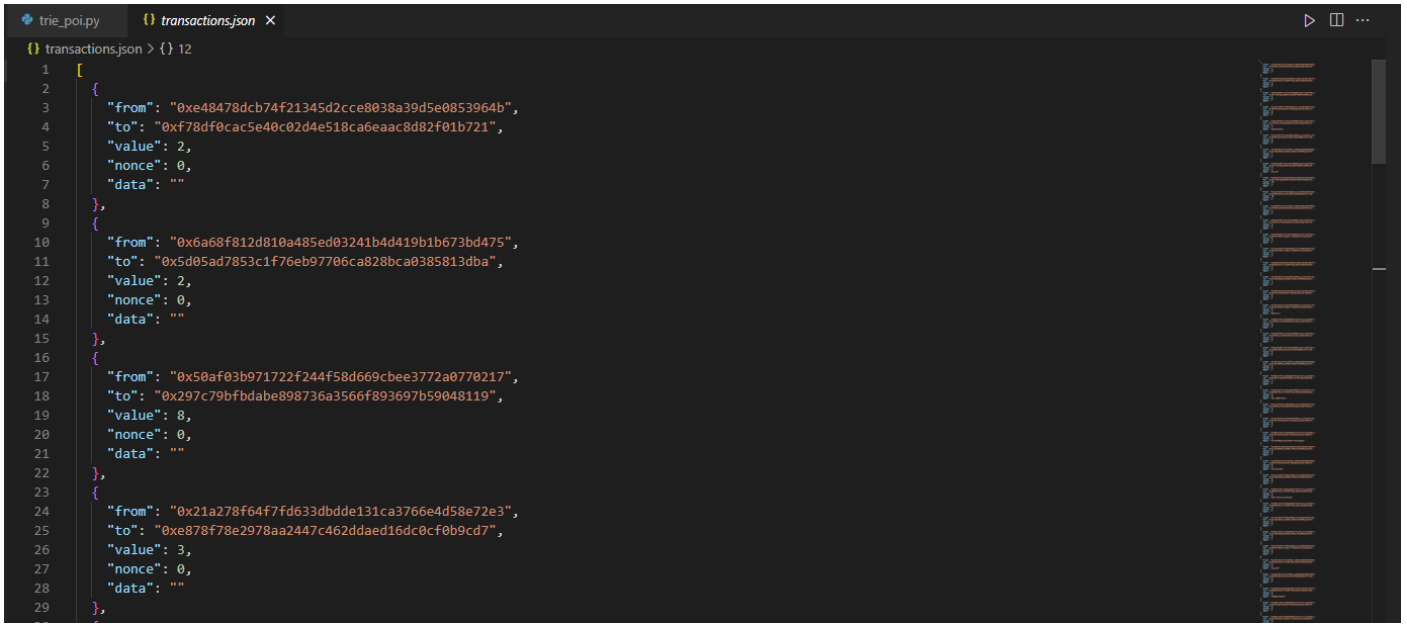
B. Outputs and Screenshots

```
senders=6, receivers=8, unique_addresses=14, total_txs=42
transactionsRoot: 32df2e105a7327e5b20f5d9e80e37d8903697b855bbbc145a84a9bdbe1f29318
Max trie depth: 8
Root changed after single-tx mutation: True
Proof for index 3:
```

1. **Initial Trie Construction**
   - The trie was built from 42 transactions.
   - The root hash was computed and printed.

2. **Root Change after Single Transaction Mutation**
   - After modifying just one transaction value, the trie root changed.
   - This proves the root is sensitive to even tiny changes.

3. **Valid Proof of Inclusion**

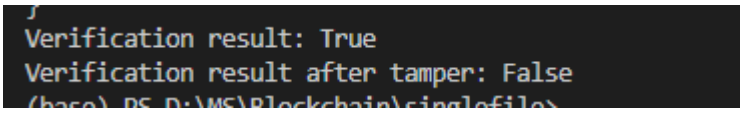


- A proof was generated for the transaction at index 3.
- Verification returned True, confirming inclusion.

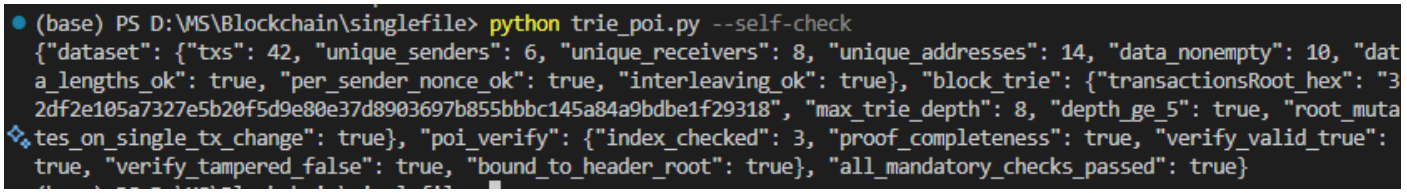
4. **Invalid Proof after Tampering**

- The proof was modified by flipping a byte.

- Verification returned False, showing that invalid proofs are rejected.



5. **Self-Check Verification**



{"dataset":
{"txs": 42, "unique_senders": 6,
"unique_receivers": 8,
"unique_addresses": 14,
"data_nonempty": 10,
"data_lengths_ok": true,
"per_sender_nonce_ok": true,
"interleaving_ok": true},

"block_trie": {"transactionsRoot_hex":"32df2e105a7327e5b20f5d9e80e37d8903697b855bbbc145a84a9bdbe1f29318",
"max_trie_depth": 8,
"depth_ge_5": true,
"root_mutates_on_single_tx_change": true},
"poi_verify": {"index_checked": 3,
"proof_completeness": true,
"verify_valid_true": true,
"verify_tampered_false": true,
"bound_to_header_root": true},
"all_mandatory_checks_passed": true}

The self-check passed, confirming all dataset, trie, and PoI requirements

C. Discussion

1. **Why PoI is efficient for light clients:**
   Light clients don't need to download the full block or all transactions. They only need:

- The block's root hash.
- A small proof that shows the path to one transaction.

  Since the proof is short, checking it is much faster and cheaper than rebuilding the whole trie. This helps devices with less storage or slower networks still verify transactions.

2. **Why keys use nibble paths:**
   Keys are split into nibbles (4-bit pieces) so the trie can branch at half-byte steps. This makes the trie more compact and avoids confusion in prefixes. If full bytes were used instead, paths would be longer and less space-efficient.