Aplikasi Pohon dalam Optimasi Sistem Teknologi Blockchain

Derwin Rustanly - 13522115[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522115@std.stei.itb.ac.id

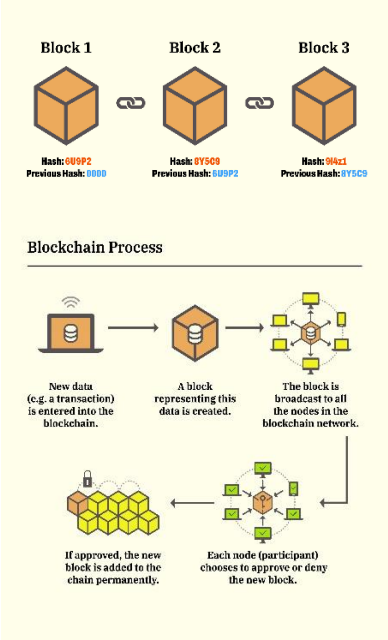
***Abstraksi*—Makalah ini bertujuan untuk menganalisis dan mengimplementasikan optimasi dalam teknologi blockchain melalui pemanfaatan struktur data pohon, khususnya pohon Merkle. Pohon Merkle akan diaplikasikan untuk mengoptimalkan validasi data transaksi serta pengelolaan transaksi secara sekaligus di dalam satu simpul (blok) dalam rantai blockchain. Dengan pendekatan ini, diharapkan adanya peningkatan efisiensi dan keamanan sistem blockchain dengan pemanfaatan struktur data pohon.**

Kata kunci – Blockchain, Optimasi. Pohon, Pohon Merkle

I. PENDAHULUAN

Seiring dengan kemajuan pesat Revolusi Industri 5.0, perkembangan teknologi telah menimbulkan transformasi yang berdampak signifikan terrhadap beberapa aspek kehidupan di masyarakat, salah satunya pada aspek keuangan. Hal ini tercermin dalam pergeseran paradigma keuangan tradisional menuju model digital yang bersifat lebih terdesentralisasi dan inovatif. Dalam hal ini, sistem teknologi blockchain sebagai salah satu implementasi sistem keuangan terdesentralisasi menawarkan aspek keamanan dan transparansi dalam penyimpanan dan pengelolaan data transaksi mata uang digital (*cryptocurrency*), seperti Bitcoin dan Ethereum dengan memanfaatkan algoritma-algoritma kriptografi.

Pada dasarmya blockchain direpresentasikan sebagai suatu struktur data graf asiklik berarah yang mencatat transaksi secara kronologis melalui serangkaian blok yang terhubung. Setiap blok dalam rantai blockchain menyimpan dan mengenkripsi informasi suatu transaksi dalam suatu nilai *hash* kriptografis, serta juga menyimpan nilai *hash* dari blok pendahulunya. Selain itu, blockchain sebagai suatu sistem keuangan yang terdesentralisasi menerapkan jaringan antar klien (*peer-to-peer network*), dalam artian setiap penambahan blok pada blockchain perlu divalidasi oleh seluruh klien pada jaringan antar klien blockchain hingga tercapai adanya suatu konsensus. Dengan pendekatan ini, teknologi blockchain dapat memastikan adanya jaminan keamanan yang tinggi karena apabila terdapat perubahan terhadap suatu blok, seluruh rantai blok berikutnya juga akan mengalami perubahan secara konsekuen, sehingga menimbulkan adanya potongan rantai blok yang invalid dan akan ditolak oleh jaringan antar klien.



Gambar I.1 Ilustrasi Cara Kerja Sistem Blockchain
Sumber: https://money.com/what-is-blockchain/

Pada makalah ini, akan dianalisis dan diimplementasikan aplikasi dari struktur data pohon, yakni pohon Merkle untuk mengoptimasi sistem teknologi blockchain, khususnya dalam hal validasi data transaksi serta pengelolaan beberapa transaksi sekaligus di dalam 1 simpul (blok) dalam rantai blockchain.

II. LANDASAN TEORI

Sebagaimana telah diuraikan pada bagian I, blockchain merupakan salah satu representasi dari graf yang memanfaatkan algoritma kriptografi, sehingga untuk menganalisisnya diperlukan pemahaman mendasar terhadap konsep graf dan kriptografi yang akan diuraikan terlebih dahulu berturut-turut pada bagian A dan B.

A. Graf

Graf merupakan suatu representasi dari kumpulan objek diskrit dan relasi yang terdapat di antara objek-objek tersebut. Suatu graf memiliki 2 komponen utama, yakni simpul (*vertex*) dan sisi (*edges*). Suatu graf didefinisikan sebagai suatu tuple berelemen 2 dengan persamaan matematis berikut.

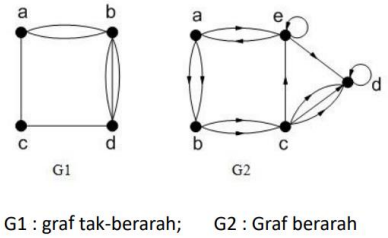
$$G = (V, E)$$

dengan V yang merupakan himpunan simpul yang terdapat pada graf dan E yang merupakan himpunan sisi yang menghubungkan sepasang simpul, atau dapat dituliskan sebagai

$$V = \{v_1, v_2, .., v_n\}$$
$$E = \{e_1, e_2, .., e_n\}$$

Karena luasnya cakupan pokok pembahasan tentang graf, pada bagian ini hanya akan dibahas beberapa konsep dan terminologi graf yang berkaitan dengan implementasi dari pohon dan blockhain, yakni sebagai berikut.

1. **Graf tak berarah** (*directed graph*). Suatu graf dikategorikan sebagai graf tak berarah apabila setiap sisinya tidak memiliki orientasi arah.
2. **Graf berarah** (*directed graph*). Suatu graf dikategorikan sebagai graf berarah apabila terdapat orientasi darah pada setiap sisinya.



G1 : graf tak-berarah; G2 : Graf berarah

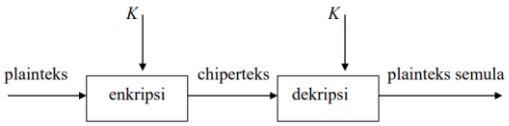
Gambar II.1 Perbedaan Graf Berarah dan Graf Tak Berarah
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

3. **Ketetanggaan** (*adjacent*)**.** Dua simpul v_i dan v_j pada suatu graf dikatakan bertetangga apabila terdapat suatu sisi yang menghubungkan kedua simpul tersebut, yakni $e = (i, j)$.
4. **Bersisian** (*incident*)**.** Sembarang sisi $e = (i, j)$ dikatakan bersisian dengan simpul v_i dan v_j.
5. **Derajat** (*degree*)**.** Derajat merupakan jumlah sisi yang bersisian pada suatu simpul. Derajat dinotasikan sebagai $d(v)$.
6. **Lintasan** (*path*)**.** Lintasan merupakan suatu barisan berselang-seling dari simpul-simpul dan sisi-sisi yang menghubungkan suatu simpul awal dengan simpul tujuan.
7. **Sirkuit** (*circuit*)**.** Sirkuit merupakan suatu lintasan yang berawal dan berakhir pada simpul yang sama.
8. **Keterhubungan** (*connectivity*)**.** Dua simpul v_i dan v_j pada suatu graf dikatakan terhubung apabila terdapat lintasan yang menghubungkan kedua simpul tersebut. Suatu graf dikategorikan sebagai graf terhubung apabila setiap pasangan simpul di graf tersebut terhubung.
9. **Upagraf** (*subgraph*)**.** Suatu graf $G' = (V', E')$ dikatakan sebagai upagraf dari graf $G = (V, E)$ jika himpunan simpul dan sisi pada graf G' merupakan himpunan bagian dari himpunan simpul dan sisi pada graf G, atau dapat dituliskan dengan notasi matematis $V' \subseteq V$ dan $E' \subseteq E$.

B. Kriptografi

Secara etimologis, kriptografi berasal dari bahasa Yunani yakni dari kata *kryptos* yang berarti tersembunyi atau rahasia dan *graphein* yang berarti menulis, yang apabila digabungkan maknanya menjadi menulis tersembunyi (*secret writing*). Secara umum, kriptografi dapat dimaknai sebagai salah satu cabang ilmu yang membahas cara-cara untuk menjaga keamanan pesan dengan cara menyandikannya menjadi bentuk lain yang tak bermakna, dengan tujuan agar pesan yang sifatnya rahasia tidak dapat dibaca dan disalahgunakan oleh pihak ketiga yang tidak berwenang. Beberapa terminologi dasar pada kriptografi adalah sebagai berikut.

1. **Pesan** (*Plaintext*). Suatu data atau informasi yang dapat dibaca dan dipahami maknanya.
2. **Cipherteks** (*Ciphertext*). Pesan yang telah disandikan sehingga tidak dapat dipahami maknanya.
3. **Enkripsi** (*Encryption*). Proses menyandikan pesan menjadi cipherteks.
4. **Dekripsi** (*Decryption*). Proses menerjemahkan cipherteks menjadi pesan semulanya.



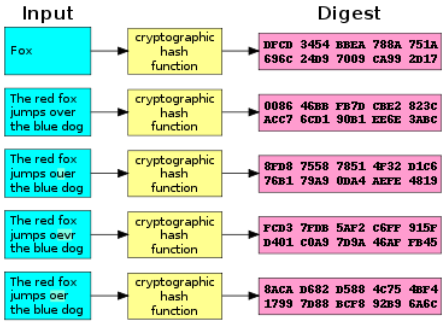
Gambar II.2 Ilustrasi Proses Enkripsi dan Dekripsi pada Kriptografi
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/16-Teori-Bilangan-Bagian3-2023.pdf

Salah satu jenis algoritma kriptografi yang diimplementasikan pada sistem blockchain adalah fungsi hash kriptografi (*CHF – Cryptographic Hash Function*). Pada dasarnya, sebuah fungsi hash adalah suatu bentuk pemetaan data (kunci) terhadap suatu lokasi di memori (nilai) sehingga pengaksesan terhadap data dapat berjalan dengan sangkil dan mangkus. Beberapa terminologi yang berkaitan dengan fungsi hash adalah sebagai berikut.

1. **Kunci** (*key*). Data atau informasi yang dimasukkan ke dalam fungsi hash untuk diolah dan dihasilkan nilai hash-nya.
2. **Nilai hash** (*value/digest*). Hasil keluaran dari fungsi hash setelah memproses kunci.
3. **Kolisi** (*collision*). Kondisi ketika terdapat dua kunci berbeda memiliki nilai hash yang sama.

Dalam hal ini, fungsi hash kriptografi memetakan beberapa informasi dengan jumlah sembarang ke suatu string dengan panjang yang tetap dengan sifat yang determistik dan satu arah. Adapun, konsep deterministik yang dimaksud merujuk pada string yang dihasilkan oleh CHF yang sifatnya konsisten untuk setiap masukan informasi, sehingga setiap informasi (kunci) yang sama akan mendapatkan string (nilai) yang sama, dan kemungkinan terjadinya suatu benturan data (kolisi) sangatlah minim. Selain itu, sifat satu arah (*irreversible*) bermakna bahwa proses dekripsi nilai hash menjadi kunci sangatlah sulit dan tidak memungkinkan untuk direalisasikan. Beberapa CHF yang umum digunakan, di antaranya SHA-256 (menghasilkan nilai hash dengan ukuran 256 bit atau 32 byte), dan SHA-512

(menghasilkan nilai hash dengan ukuran 512 bit atau 64 byte).

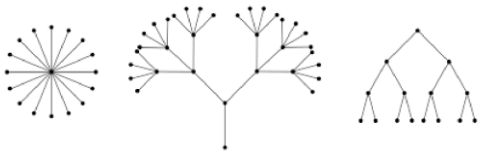


Gambar II.3 Ilustrasi Fungsi Hash Kriptografis (CHF)
Sumber:
https://en.wikipedia.org/wiki/Cryptographic_hash_function

C. Pohon

Secara sederhana, struktur data pohon dapat didefinisikan sebagai suatu graf yang tak berarah, terhubung, dan tidak memiliki sirkuit. Berikut merupakan beberapa karakteristik yang juga menggambarkan definisi matematis dari pohon.

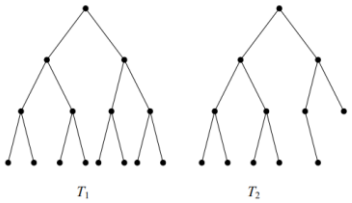
1. Suatu graf tak berarah $G = (V, E)$ dengan jumlah sisi m dan jumlah simpul n adalah pohon.
2. Setiap pasangan simpul G terhubung.
3. G merupakan graf terhubung dengan selisih jumlah sisi dan jumlah simpul sebesar 1 ($m = n - 1$).
4. G tidak mengandung sirkuit
5. Penambahan satu sisi pada G akan membentuk sirkuit.



Gambar II.4 Ilustrasi Struktur Data Pohon
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/22-Pohon-Bag1-2023.pdf

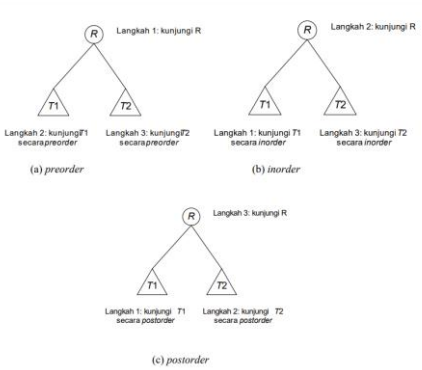
Beberapa terminologi dari pohon yang akan mendasari analisis pada makalah ini, antara lain sebagai berikut.

1. **Pohon berakar** (*rooted tree*). Pohon yang salah satu simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah (pada umumnya, tanda panah pada sisi diabaikan).
2. **Induk** (*parent*). Sebuah simpul dalam pohon yang memiliki simpul-simpul lain yang terhubung secara langsung kepadanya, atau dapat dikatakan sebagai simpul yang lebih tinggi dalam hierarki.
3. **Anak** (*child*). Sebuah simpul dalam pohon yang terhubung secara langsung ke simpul induk, atau dapat dikatakan sebagai simpul yang lebih rendah dalam hierarki.
4. **Saudara kandung** (*sibling*). Pasangan simpul di dalam pohon dikatakan bersaudara apabila kedua simpul tersebut terhubung kepada simpul induk yang sama.
5. **Upapohon** (*subtree*). Suatu pohon $P' = (V', E')$ dikatakan sebagai upapohon dari pohon $P = (V, E)$ jika seluruh himpunan simpul dan sisi pada pohon P' merupakan himpunan bagian dari himpunan simpul dan sisi pada pohon P.
6. **Derajat** (*degree*). Jumlah upapohon (atau jumlah anak) pada suatu simpul.
7. **Daun** (*leaf*). Simpul yang memiliki derajat 0.
8. Tingkat/aras (*level*). Panjang lintasan dari akar ke suatu simpul tertentu.
9. **Kedalaman** (*depth*). Tingkat maksimum dalam suatu pohon.
10. **Pohon terurut** (*ordered tree*). Pohon berakar yang urutan anak-anaknya penting.
11. **Pohon n-ary** (*n-ary tree*). Pohon berakar yang setiap simpul cabangnya berderajat maksimal n.
12. **Pohon biner** (*binary tree*). Pohon berakar yang setiap simpul cabangnya berderajat maksimal 2. Ketiga komponen utamanya meliputi akar, upapohon kiri, dan upapohon kanan.
13. **Pohon biner seimbang** (*b-tree*). Pohon biner dengan perbedaan kedalaman upapohon kiri dan kanan maksimal 1.



Gambar II.5 Ilustrasi Pohon Biner Seimbang
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/22-Pohon-Bag2-2023.pdf

14. **Penelusuran** (*traversal*). Proses mengunjungi dan memproses setiap simpul dalam pohon dengan 3 metode utama, yaitu sebagai berikut.
    a. **Preorder**. Mengunjungi simpul akar, dilanjutkan dengan subpohon kiri dan kanan secara berurutan dan rekursif.
    b. **Inorder**. Mengunjungi subpohon kiri, dilanjutkan dengan mengunjungi akar dan upapohon kanan secara berurutan dan rekursif.
    c. **Postorder**. Mengunjungi subpohon kiri, dilanjutkan dengan subpohon kanan dan simpul akar secara berururtan dan rekursif.



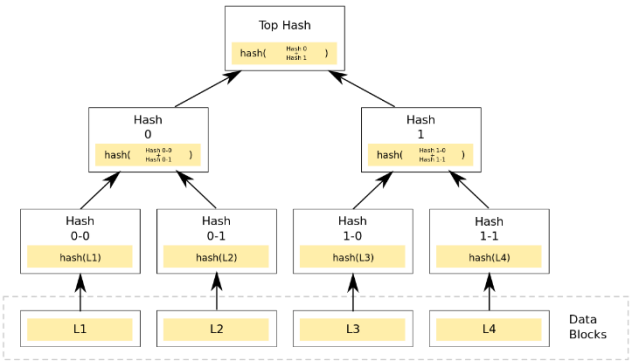
Gambar II.6 Ilustrasi Proses Traversal pada Pohon

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/23-Pohon-Bag2-2023.pdf

D. Pohon Merkle

Pohon Merkle merupakan salah satu implementasi dari pohon biner seimbang yang ditemukan oleh ilmuwan Ralph Merkle pada tahun 1980-an, yang dirancang untuk memungkinkan proses verifikasi integritas data dengan efisien. Pada dasarnya, pohon Merkle melibatkan penggunaan fungsi hash kriptografis (CHF) untuk membentuk pohon hash biner (*binary hash tree*) dari sejumlah data, dimulai dari simpul-simpul daun (*leaf*) yang mewakili potongan-potongan data individual. Algoritma pembentukan Pohon Merkle adalah sebagai berikut.

1. Lakukan pemetaan pada data yang akan direpresentasikan oleh pohon Merkle. Apabila jumlah data tidaklah genap, duplikasikan datum terakhir agar jumlah data menjadi genap.
2. Tempatkan nilai hash keseluruhan data pada simpul daun pohon Merkle.
3. Pasangkan 2 simpul yang merepresentasikan nilai hash, dan lakukan pemetaan pada hasil konkatenasi dari nilai hash dari kedua simpul tersebut.
4. Ulangi langkah ke-3 secara rekursif, hingga tersisa 1 nilai hash yang merepresentasikan simpul akar dari pohon Merkle.



Gambar II.7 Visualisasi Algoritma Pembentukan Pohon Merkle
Sumber: https://en.wikipedia.org/wiki/Merkle_tree

Pohon Merkle berguna untuk memvalidasi salah satu bagian data tertentu tanpa perlu menyimpan dan mengakses keseluruhan data. Pada contoh implementasi dari gambar II.8, apabila akan dilakukan validasi terhadap data L2, hanya perlu dilakukan pengelompokan nilai hash pada data L2 (simpul 0-1) dengan nilai hash pada data L1 (simpul 0-0). Kemudian dilakukan penyandian (*hashing*) pada data tersebut (simpul 0), lalu dikontatenasikan lagi dengan nilai hash dari pasangan data lainnya secara rekursif, dalam hal ini nilai hash dari hasil konkatenasi nilai hash data L3 dan L4 (simpul 1), hingga menyisakan nilai hash yang mewakili simpul akar dari pohon Merkle. Apabila dilakukan perubahan terhadap data L2, nilai hash yang didapatkan pada simpul akar juga akan berubah. Perubahan simpul akar yang tidak terotentikasi mengindikasikan adanya salah satu bagian data yang tidak valid.

III. Analisis dan Implementasi

Pada bagian ini, akan diuraikan mengenai analisis dan implementasi pohon untuk mengoptimasi sistem blockchain dalam hal validasi data transaksi serta pengelolaan blok untuk menyimpan berbagai data transaksi dalam suatu simpul pada rantai blockchain.

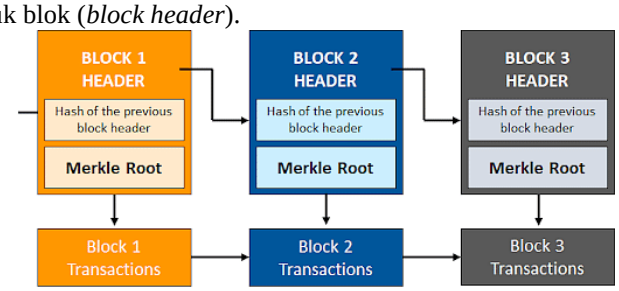
A. Analisis Optimasi Sistem Blockchain

Sistem Blockchain merupakan suatu struktur data terdesentralisasi yang merepresentasikan transaksi secara kronologis melalui serangkaian blok yang terhubung. Setiap blok dalam rantai blockchain menyimpan informasi transaksi yang telah dienkripsi dengan menggunakan fungsi hash kriptografis (CHF), dan terkait dengan blok pendahulunya melalui nilai hash dari blok tersebut.

Secara naif, suatu simpul (blok) pada rantai blockhain dapat dikatakan hanya dapat mewakili satu jenis transaksi tertentu, sehingga proses validasi suatu transaksi yang dilakukan oleh jaringan antar klien (*peer-to-peer network*) dapat dinyatakan kurang optimal karena setiap klien harus memiliki salinan utuh dari keseluruhan rantai blok untuk dapat mendeteksi adanya nilai hash pada salah satu blok yang tidak valid.

Pohon Merkle dapat diimplementasikan pada sistem Blockchain untuk mengoptimasikan proses validasi dan verifikasi terhadap serangkaian data transaksi yang disimpan di dalam suatu blok. Ide utama dari pendekatan optimasi ini adalah dengan mengubah konsep pengelolaan blok dalam rantai blockchain yang semulanya hanya merepresentasikan satu jenis transaksi menjadi serangkaian transaksi yang dikelompokkan berdasarkan kesamaan tertentu, umumnya berupa kesamaan waktu dilakukannya proses transaksi.

Sebagai contoh, apabila terdapat transaksi yang dilakukan oleh pengguna A dan pengguna B di dalam waktu yang bersamaan, kedua transaksi tersebut dapat direpresentasikan sebagai satu blok pada rantai blockchain. Proses penggabungan sekumpulan transaksi yang dilakukan pada waktu yang sama diimplementasikan dengan algoritma pembentukan pohon Merkle yang telah diuraikan sebelumnya pada bagian D bab II. Setelah didapatkan simpul akar dari pohon Merkle yang mewakili kombinasi nilai hash dari keseluruhan transaksi, nilai hash tersebut akan dikombinasikan dengan nilai hash dari blok pendahulunya untuk dilakukan pemetaan dan disimpan sebagai nilai hash pada blok yang bersesuaian, atau disebut juga sebagai tajuk blok (*block header*).



Merkle Tree breaking the data into tiny parts of information

Gambar III.1 Visualisasi Optimasi Sistem Blockchain
Sumber :
https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain#cryptographic_hash_functions

Adapun, optimasi proses validasi terhadap integritas dari suatu transaksi tertentu dapat diimplementasikan dengan cara mendapatkan nilai hash dari tajuk blok (*block header*) yang merepresentasikan serangkaian transaksi yang juga dilakukan pada waktu yang bersamaan dengan data transaksi yang ingin divalidasi. Dengan demikian, jaringan antar klien tidak perlu menyimpan salinan dari keseluruhan rantai blockchain untuk mendeteksi adanya ketidakvalidan salah satu transaksi tertentu dan proses validasi dan verifikasi transaksi dapat berjalanan dengan lebih sangkil dan mangkus.

B. Implementasi Sistem Blockchain

Sebagai bahan eksperimen, optimasi dari sistem blockchain yang diuraikan pada bagian A telah diimplementasikan dengan bahasa pemrograman C dalam bentuk ADT (*Abstract Data Type*), dengan struktur program sebagai berikut.

1. **blockchain.h**. Berisi deklarasi tipe serta definisi fungsi dan prosedur ADT Blockchain dengan representasi graf berarah asiklik (*directed acyclic graph*).
2. **blockchain.c**. Berisi realisasi dari fungsi dan prosedur dari ADT Blockchain.
3. **tree.h**. Berisi deklarasi tipe serta definisi fungsi dan prosedur ADT MerkleTree dengan representasi pohon biner seimbang (*b-tree*).
4. **tree.c**. Berisi realisasi dari fungsi dan prosedur dari ADT MerkleTree.
5. **sha256.h**. Berisi deklarasi tipe serta definisi fungsi dan prosedur modul fungsi hash kriptografis (CHF) dengan jenis SHA-256.
6. **sha256.c**. Berisi realisasi dari fungsi dan prosedur modul fungsi hash kriptografis (CHF) dengan jenis SHA-256.

Berikut ini merupakan sketsa struktur data yang telah diimplementasikan pada file blockchain.h.

```c
typedef struct block* Address;
typedef struct block {
    ListData list;
    int NEff;
    data hash;
    Address prev;
} block;

#define LIST(p) (p)->list
#define NEFF(p) (p)->NEff
#define BHASH(p) (p)->hash
#define PREV(p) (p)->prev

typedef struct Blockchain{
    Address last;
    int length;
} Blockchain;

#define LAST(p) (p).last
#define len(p) (p).length
```

```c
void createBlockchain(Blockchain *B);
boolean isEmpty(Blockchain B);
Address newBlock();
void insertBlock(Blockchain* B);
void displayList(ListData list, int n);
void displayBlockchain(Blockchain B);
```

Gambar III.2 ADT Blockchain dalam bahasa C
Sumber: Dokumen Penulis

ADT Blockchain memiliki beberapa primitif yang berupa fungsi dan prosedur, di antaranya prosedur createBlockchain yang berfungsi sebagai konstruktor untuk membentuk blockchain kosong, predikat isEmpty yang merupakan fungsi untuk mengecek apakah suatu blockchain bersifat kosong, fungsi newBlock yang mengembalikan alamat (*address*) terhadap suatu simpul (blok) yang dibentuk berdasarkan sekumpulan transaksi tertentu, prosedur insertBlock untuk menambahkan blok ke dalam rantai blockchain, prosedur displayList untuk menampilkan rangkaian transaksi yang terdapat di dalam suatu blok, serta displayBlockchain untuk menampilkan keseluruhan blok dalam suatu blockchain.

Berikut ini merupakan sketsa struktur data yang telah diimplementasikan pada file tree.h.

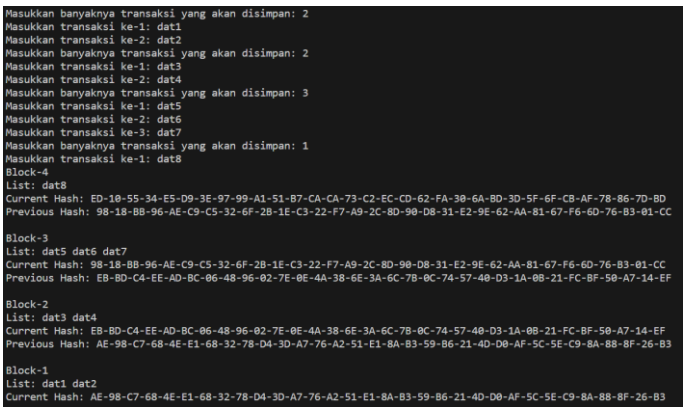
```c
typedef BYTE* data;
typedef data* ListData;

typedef struct treeNode* TAddress;
typedef struct treeNode {
    data info;
    BYTE hash[32];
    TAddress left;
    TAddress right;
} TreeNode;

#define INFO(p) (p)->info
#define HASH(p) (p)->hash
#define LEFT(p) (p)->left
#define RIGHT(p) (p)->right

typedef TAddress MTree;
void hash(data string, BYTE hash[32]);
void printHash(BYTE hash[32]);

TAddress createMNode(data string);
void buildMTree(ListData list, int start, int end, MTree *p);
void printPostOrder(MTree p);
```

Gambar III.3 ADT MerkleTree dalam bahasa C
Sumber: Dokumen Penulis

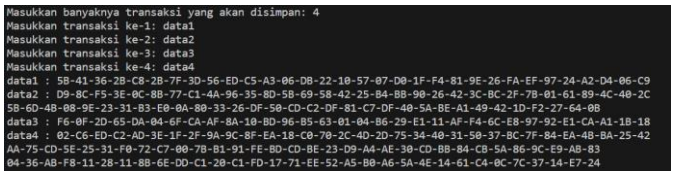
ADT MerkleTree memiliki beberapa primitif dasar yang meliputi prosedur hash yang memetakan suatu string dengan panjang sembarang ke nilai hash berukuran 256 bit, prosedur printHash yang berfungsi untuk menampilkan suatu nilai hash tertentu, fungsi createMNode yang mengembalikan alamat (*address*) dari suatu simpul pohon, prosedur buildMTree yang berfungsi sebagai konstruktor yang membentuk pohon merkle dari serangkaian data tertentu, serta prosedur printPostOrder yang berfungsi untuk menampilkan keseluruhan simpul pada pohon Merkle secara postorder.

Adapun algoritma fungsi hash kriptografis SHA-256 yang digunakan telah disusun dan diimplementasikan oleh Brad Conte (sumber: https://github.com/B-Con/crypto-algorithms).

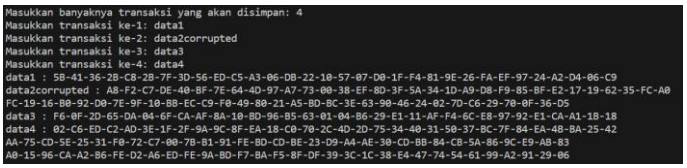
Hasil implementasi dari program optimasi sistem blockchain yang telah dirancang adalah sebagai berikut.



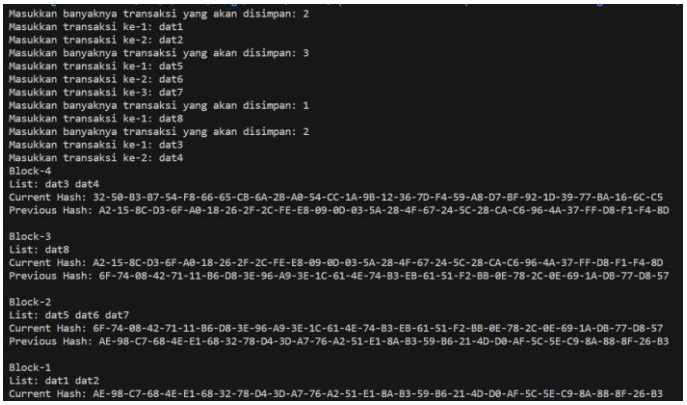
Gambar III.4 Implementasi Optimasi Sistem Blockchain dengan 4 simpul (blok)
Sumber: Dokumen Penulis



Gambar III.5 Implementasi Algoritma Pembentukan Pohon Merkle yang Ditampilkan Secara Postorder
Sumber: Dokumen Penulis



Gambar III.6 Implementasi Validasi Data Transaksi (Perubahan terhadap data2 menyebabkan nilai hash pada simpul akar yang juga berubah)
Sumber: Dokumen Penulis



Gambar III.6 Implementasi Validasi Data Transaksi (Perubahan orientasi pada data masukan di blok tertentu menyebabkan perubahan pada nilai hash blok yang bersangkutan)
Sumber: Dokumen Penulis

Implementasi dari keseluruhan program optimasi sistem blockchain dapat diakses pada laman berikut https://github.com/DerwinRustanly/Aplikasi-Pohon-Dalam-Optimasi-Sistem-Teknologi-Blockchain.

IV. KESIMPULAN

Berdasarkan implementasi dari konsep optimasi sistem blockchain yang memanfaatkan aplikasi dari pohon, dapat disimpulkan bahwa pohon Merkle sebagai pohon hash kriptografis dapat mengoptimasi proses validasi transaksi dengan cara membentuk suatu nilai hash yang merepresentasikan beberapa data transaksi yang dilakukan dalam waktu yang bersamaan. Adapun, model dari sistem blockchain teroptimasi yang direpresentasikan dengan ADT Blockchain merupakan bentuk implementasi sederhana yang cukup menggambarkan fitur dan mekanisme dari sistem blockchain yang sesungguhnya.

V. PENUTUP

Akhir kata, penulis memanjatkan puji syukur kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, proses pembuatan makalah Matematika Diskrit dapat terselesaikan dengan baik. Selain itu, penulis juga ingin mengucapkan terima kasih kepada Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T. selaku dosen kelas 02 Matematika Diskrit yang telah memaparkan ilmu dan konsep Matematika Diskrit yang dapat dipahami secara mendalam oleh penulis, serta kepada Bapak Dr. Ir. Rinaldi Munir, M.T. yang telah mewadahi dan memfasilitasi berbagai sarana pembelajaran mata kuliah Matematika Diskri dalam bentuk sumber belajar utama maupun referernsi penunjang lainnya. Penulis berharap makalah ini dapat mendatangkan manfaat khususnya dalam eksplorasi terkait aplikasi pohon dalam optimasi sistem teknologi blockchain, serta dapat dijadikan rujukan bagi penelitian berikutnya.

DAFTAR PUSTAKA

[1] https://aws.amazon.com/id/what-is/blockchain/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc, diakses pada 9 Desember 2023 pukul 12.00
[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/23-Pohon-Bag2-2023.pdf, diakses pada 9 Desember 2023 pukul 19.00
[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/22-Pohon-Bag1-2023.pdf, diakses pada 9 Desember 2023 pukul 19.00
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 9 Desember 2023 pukul 19.00
[5] https://www.researchgate.net/publication/358740207_An_Overview_of_Trees_in_Blockchain_Technology_Merkle_Trees_and_Merkle_Patricia_Tries, diakses pada 9 Desember 2023 pukul 20.00
[6] https://github.com/B-Con/crypto-algorithms, diakses pada 9 Desember 2023 pukul 20.00
[7] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2022-2023/Makalah2022/Makalah-Matdis-2022%20(160).pdf, diakses pada 9 Desember 2023 pukul 20.00

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2023

LAMPIRAN

Realisasi Fungsi dan Prosedur ADT Blockchain dalam Bahaasa C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "blockchain.h"

void createBlockchain(Blockchain *B){
   LAST(*B) = NULL; len(*B) = 0 ;
}
boolean isEmpty(Blockchain B){
   return (LAST(B) == NULL);
}

Address newBlock(){
   int n; data temp; ListData list; Address p;
   p = (Address) malloc (sizeof(block));
   if(p != NULL){
      printf("Masukkan banyaknya transaksi yang akan disimpan: ");
      scanf("%d", &n);
      LIST(p) = (ListData) malloc (n*sizeof(data));
      if(LIST(p) != NULL){
         for(int i = 0; i < n; i++){
            temp = (data) malloc (32*sizeof(BYTE));
            printf("Masukkan transaksi ke-"); printf("%d", i+1); printf(": ");
            scanf("%s", temp);
            LIST(p)[i] = temp;
         }
      }
      PREV(p) = NULL;
      NEFF(p) = n;
   }
}

void displayList(ListData list, int n){
   for(int i = 0; i < n; i++){
      printf("%s ", list[i]);
   }
   printf("\n");
}

void insertBlock(Blockchain* B){
   Address p; TAddress t;
   p = newBlock();
   buildMTree(LIST(p), 0, NEFF(p)-1, &t);
   // printPostOrder(t);
   PREV(p) = LAST(*B);
   HASH(p) = (data) malloc (256*sizeof(BYTE));
   if(!isEmpty(*B)){
      data concatHash = (data)malloc(64*sizeof(BYTE));
      memcpy(concatHash, HASH(t), 32);
      memcpy(concatHash + 32, BHASH(PREV(p)), 32);
      hash(concatHash, BHASH(p));
   } else{
      strcpy(BHASH(p), HASH(t));
   }
   LAST(*B) = p;
   len(*B)++;
}

void displayBlockchain(Blockchain B){
   Address p = LAST(B); int n = len(B);
   while(p != NULL){
      printf("Block-%d\n", n);
      printf("List: ");
      displayList(LIST(p), NEFF(p));
      printf("Current Hash: ");
      printHash(BHASH(p));
      if(PREV(p) != NULL){
         printf("Previous Hash: ");
         printHash(BHASH(PREV(p)));
      }
      printf("\n");
      p = PREV(p); n--;
   }
}
```

Realisasi Fungsi dan Prosedur ADT Tree dalam Bahaasa C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sha256.h"
#include "tree.h"

void printHash(BYTE hash[32]){
   for (int i = 0; i < 32; i++)
   {
      printf("%02X", hash[i]);
      if (i < 31)
         printf("-");
   }
   printf("\n");
}

void hash(data string, BYTE hash[32]){
   SHA256_CTX ctx;
   sha256_init(&ctx);
   sha256_update(&ctx, string, strlen(string));
   sha256_final(&ctx, hash);
}

TAddress createMNode(data string) {
   TAddress node = (TAddress) malloc(sizeof(TreeNode));
   INFO(node) = string;
   hash(string, HASH(node));
   LEFT(node) = NULL;
   RIGHT(node) = NULL;
```

```
    return node;
}

void buildMTree(ListData list, int start, int end, MTree *p) {
    if (start == end) {
        *p = createMNode(list[start]);
    } else{
    int mid = (start + end) / 2;
    TAddress leftNode; TAddress rightNode;
    buildMTree(list, start, mid, &leftNode);
    buildMTree(list, mid + 1, end, &rightNode);
        data concatHash = (data)malloc(64*sizeof(BYTE));
        memcpy(concatHash, HASH(leftNode), 32);
        memcpy(concatHash + 32, HASH(rightNode), 32);
        TAddress parentNode = createMNode(concatHash);
        LEFT(parentNode) = leftNode;
        RIGHT(parentNode) = rightNode;
        *p = parentNode;
        free(concatHash);
    }
}

void printPostOrder(MTree p){
    if(p != NULL){
        if(LEFT(p) == NULL && RIGHT(p) == NULL)
            printf("%s : ", INFO(p));
        printPostOrder(LEFT(p));
        printPostOrder(RIGHT(p));
        printHash(HASH(p));
    }
}
```