**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**NGUYỄN PHÚC THUẦN – 20120380**

**TRẦN NAM TUẤN – 20120615**

**NGUYỄN TẠ HUY HOÀNG – 20120482**

**NGUYỄN TIẾN ĐẠT – 20120267**

**BÁO CÁO ĐỒ ÁN**

**SIMPLIFIED BLOCKCHAIN IMPLEMENTATION AND VERIFICATION**

**| Giáo viên hướng dẫn |**

**ThS. NGÔ ĐÌNH HY**

**PGS. TS. NGUYỄN ĐÌNH THÚC**

**Chuyên ngành: An Toàn Thông Tin**

Thành phố Hồ Chí Minh – 2023

|  |
| --- |
| MỤC LỤC |

1. **Blockchain**
2. Cấu trúc Blocks
3. Cấu trúc Blockchain
4. **Merkle Tree**
5. Cấu trúc cây Merkle
6. Xây dựng một cây Merkle trong Blockchain.
7. **Command Line Client.**
8. **Blockchain**

* Blockchain không chỉ là một bộ dữ liệu phân tán, mà còn là một công nghệ đặc biệt với nhiều tính năng và đặc điểm quan trọng. Một trong những điểm quan trọng nhất của blockchain là tính an toàn và bảo mật. Dữ liệu trên blockchain được lưu trữ trong các khối (blocks) liên kết với nhau thông qua các hàm mã hóa, tạo ra một chuỗi không thể thay đổi mà không làm ảnh hưởng đến các khối trước đó. Điều này giúp ngăn chặn việc thay đổi thông tin lưu trữ và tăng tính minh bạch của dữ liệu.
* Việc blockchain là một cơ sở dữ liệu mở có nghĩa là nó được phân phối trên nhiều máy tính và không thuộc quyền kiểm soát của một tổ chức hay cá nhân cụ thể. Mọi thay đổi trong cơ sở dữ liệu phải được đồng thuận từ một phần lớn các thành viên trong mạng, tạo ra một hệ thống phi tập trung và có tính chống gian lận cao.
* Blockchain đã mở ra một loạt các ứng dụng, trong đó cryptocurrency là một trong những ứng dụng tiêu biểu nhất. Bitcoin và Ethereum là hai ví dụ nổi tiếng về tiền mã hóa được xây dựng trên nền tảng blockchain. Sự kết hợp giữa blockchain và smart contracts (hợp đồng thông minh) cũng tạo ra khả năng thực hiện các giao dịch tự động và không cần sự can thiệp của bên thứ ba, mở ra nhiều tiềm năng trong lĩnh vực tài chính, logitics, y tế, và nhiều lĩnh vực khác.

1. **Cấu trúc block**

* Trong blockchain, block chính là phần chứa những dữ liệu có giá trị. Ví dụ, các block của Bitcoin chứa các transactions (giao dịch), phần cốt yếu của bất kì cryptocurrency nào. Ngoài những giao dịch này, block còn chứa những thông tin kỹ thuật khác, ví dụ version, timestamp và hash (mã băm) của block trước.
* Cấu trúc 1 block như sau:

|  |
| --- |
| type Block struct {  **PrevBlockHash** []byte  **Transactions** []\*Transaction  **Hash** []byte  **Timestamp** int64  **MerkleRootChecksum** []byte |

* **Timestamp** là thời điểm block được tạo, **PrevBlockHash** là hash của block trước đó, **Hash** chính là hash của block này, **MerkleRootChecksum** đảm bảo tính toàn vẹn của **Block**, **Transactions** là dữ liệu các giao dịch được lưu trong block.
* Hàm tạo một Block mới:

Đây là một đoạn mã trong ngôn ngữ lập trình Go (Golang) để tạo một khối mới trong blockchain. Dưới đây là phần giải thích của hàm:

1. Khai báo hàm:

func **NewBlock**(transactions []\***Transaction**, prevBlockHash []byte) (\***Block**, error) {

Hàm **NewBlock** được khai báo với hai đối số đầu vào: transactions là một slice các con trỏ đến các đối tượng **Transaction**, và prevBlockHash là một slice byte chứa hash của khối trước đó. Hàm trả về một con trỏ đến đối tượng **Block** và một giá trị lỗi (error).

2. Khởi tạo đối tượng Block:

block := &**Block**{

**Timestamp**: time.Now().Unix(),

**Transactions**: transactions,

**PrevBlockHash**: prevBlockHash,

}

Một đối tượng **Block** mới được tạo và được gán cho biến block. Các trường của khối (thời điểm, giao dịch, và hash của khối trước đó) được khởi tạo với giá trị tương ứng.

3. Tính toán hash cho khối:

err := block.**SetHash()**

Gọi phương thức **SetHash()** của đối tượng **Block**. Phương thức này thường được sử dụng để tính toán và gán giá trị hash cho khối. Nếu có lỗi trong quá trình tính toán hash, giá trị lỗi sẽ được gán vào biến err.

4. Tính toán **Merkle Root checksum**:

block.**MerkleRootChecksum** = **NewMerkleTree**(block.**Transactions**).**GetMerkleRootChecksum()**

Tạo một cây Merkle mới từ các giao dịch trong khối sử dụng hàm **NewMerkleTree**, sau đó lấy giá trị Merkle Root checksum bằng cách gọi phương thức **GetMerkleRootChecksum()**. Kết quả được gán vào trường **MerkleRootChecksum** của khối.

5. Trả về khối và giá trị lỗi (nếu có):

return block, err

Hàm trả về đối tượng block và giá trị lỗi (nếu có). Điều này cho phép người gọi hàm kiểm tra xem việc tạo khối có thành công hay không và xử lý lỗi nếu cần thiết.

* Hashing là quá trình tạo ra một mã băm (hash) từ một dữ liệu cụ thể. Mỗi dữ liệu sẽ tương ứng với một hash duy nhất. Hàm băm (hash function) là một hàm nhận đầu vào là dữ liệu bất kỳ và trả về một hash có kích thước cố định. Các tính chất quan trọng của hàm băm bao gồm:

1. Không thể tái tạo dữ liệu gốc từ hash: Một hash không thể được sử dụng để phục hồi dữ liệu gốc. Điều này đảm bảo rằng quá trình hashing là một chiều, không thể đảo ngược.
2. Dữ liệu cố định tương ứng với một hash cố định: Một dữ liệu nhất định sẽ luôn tạo ra một hash cụ thể và không trùng lặp với hash của bất kỳ dữ liệu nào khác.
3. Thay đổi nhỏ trong dữ liệu tạo ra hash khác biệt hoàn toàn: Ngay cả việc thay đổi một bit nhỏ trong dữ liệu cũng sẽ tạo ra một hash hoàn toàn khác biệt.

* Hàm băm thường được sử dụng để kiểm tra tính toàn vẹn của dữ liệu. Ví dụ, mã hash thường được cung cấp cùng với phần mềm để kiểm tra xem phần mềm có bị chỉnh sửa hay không. Khi tải về phần mềm, bạn có thể tính toán lại hash của nó và so sánh với mã hash được cung cấp để đảm bảo tính toàn vẹn.
* Trong hệ thống blockchain, hashing đóng một vai trò quan trọng trong việc đảm bảo tính toàn vẹn của các khối. Mỗi khối có một hash được tạo ra dựa trên dữ liệu của nó và mã hash của khối liền kề trước đó. Điều này đảm bảo rằng một khi một khối thay đổi, thì cả chuỗi các khối phía sau nó cũng sẽ phải được tính toán lại, tăng cường tính bất biến của blockchain.

1. **Cấu trúc Blockchain:**

* Ở bản chất, blockchain chỉ là một cơ sở dữ liệu có một cấu trúc nhất định: nó là một danh sách có thứ tự, với các liên kết ngược. Điều này có nghĩa là các khối được lưu trữ theo thứ tự chèn và mỗi khối kết nối với khối trước đó. Cấu trúc này cho phép nhanh chóng lấy được khối mới nhất trong chuỗi và lấy một khối thông qua giá trị hash của nó.
* Xây dựng cấu trúc blockchain bằng một mảng các con trỏ. Mỗi phần tử trong mảng là một block.Vậy Blockchain là mảng của các block nối lại với nhau bằng giá trị hash.

|  |
| --- |
| type **Blockchain** struct {  blocks []\***Block**  } |

- Đoạn mã định nghĩa một kiểu dữ liệu mới trong Golang được gọi là Blockchain. Đây là một kiểu dữ liệu tùy chỉnh (custom type) được sử dụng để biểu diễn một chuỗi khối trong blockchain. Dưới đây là giải thích chi tiết về kiểu dữ liệu Blockchain:

1. type Blockchain struct: Dòng này định nghĩa một kiểu dữ liệu mới có tên là **Blockchain**. struct ở đây là một cấu trúc dữ liệu trong Golang, giúp nhóm các trường dữ liệu lại với nhau.

2. blocks []\*Block: Trong cấu trúc Blockchain, có một trường tên là blocks, và nó là một slice của các đối tượng kiểu **Block**. Mỗi phần tử trong slice này đại diện cho một khối trong chuỗi khối.

3. []\***Block**: Đây là kiểu dữ liệu của slice, một mảng động của các con trỏ đến đối tượng kiểu **Block**.

Kiểu dữ liệu **Blockchain** này được sử dụng để tạo ra các đối tượng biểu diễn một chuỗi khối trong blockchain, trong đó mỗi khối là một phần tử trong mảng blocks.

* Để thêm 1 block mới vào blockchain, ta phải cần 1 block liền kề trước nó, vì thế phải tạo 1 block đầu tiên bằng hàm newGenesisBlock

|  |
| --- |
| func **newGenesisBlock**(genesisData []byte) (\*Block, error) {  if len(genesisData) == 0 {  genesisData, \_ = **generateRandomByteArray()**  }  genesisTransaction := **Transaction**{  Data: genesisData,  }  block := &**Block**{  **Timestamp**: time.Now().Unix(),  **Transactions**: []\***Transaction**{&genesisTransaction},  **PrevBlockHash**: []byte{},  }  // Calculate and set the block's hash  err := block.**SetHash()**  return block, err  } |

* Đoạn mã này định nghĩa hàm **newGenesisBlock**, có nhiệm vụ tạo ra khối genesis, tức là khối đầu tiên trong chuỗi khối của blockchain. Dưới đây là giải thích từng phần của hàm:

1. if len(genesisData) == 0 {: Dòng mã kiểm tra nếu dữ liệu của khối genesis (genesisData) có độ dài bằng 0. Nếu đúng, tức là không có dữ liệu được cung cấp, hàm sẽ thực hiện các bước để tạo dữ liệu ngẫu nhiên cho khối genesis.

genesisData, \_ = **generateRandomByteArray()**: Gọi hàm **generateRandomByteArray()** để tạo ra một mảng byte ngẫu nhiên và gán nó vào biến genesisData. Dấu gạch dưới (\_) được sử dụng để bỏ qua giá trị lỗi trả về từ hàm này (nếu có). Điều này đảm bảo rằng genesisData sẽ chứa một giá trị ngẫu nhiên nếu không có dữ liệu được cung cấp.

2. genesisTransaction := **Transaction**{ Data: genesisData, }: Tạo một giao dịch đặc biệt cho khối genesis, được gọi là genesisTransaction. Giao dịch này có dữ liệu được chứa trong biến genesisData.

3. block := &Block{ **Timestamp**: time.Now().Unix(), **Transactions**: []\***Transaction**{&genesisTransaction}, **PrevBlockHash**: []byte{}, }: Tạo một đối tượng khối (`block`). Các thuộc tính của khối bao gồm:

- **Timestamp**: Thời điểm tạo khối, được thiết lập bằng thời điểm hiện tại ở đơn vị giây (Unix timestamp).

- **Transactions**: Một slice chứa con trỏ đến giao dịch. Trong trường này, chỉ có một giao dịch, tức là genesisTransaction.

- **PrevBlockHash**: Một slice chứa giá trị hash của khối trước đó. Vì đây là khối genesis, nó không có khối trước đó, nên **PrevBlockHash** được đặt là một slice rỗng.

4. err := block.**SetHash()**: Gọi hàm **SetHash** để tính toán và thiết lập giá trị hash cho khối. Bất kỳ lỗi nào xuất hiện trong quá trình này cũng được lưu trữ trong biến err.

5. return block, err: Trả về con trỏ đến đối tượng khối (block) và giá trị lỗi (err).

- Tiếp theo, tạo một blockchain mới với genesis block vừa tạo ở phía trên

|  |
| --- |
| func **NewBlockchain**(identifier string) (\***Blockchain**, error) {  // Create the Genesis block  genesisBlock, err := **newGenesisBlock**([]byte(identifier))  return &**Blockchain**{  **Blocks**: []\*Block{genesisBlock},  }, err  } |

* Đoạn mã này định nghĩa hàm **NewBlockchain** với mục đích tạo ra một đối tượng blockchain mới, bao gồm cả việc tạo khối genesis. Dưới đây là giải thích từng phần của mã:

1. genesisBlock, err := **newGenesisBlock**([]byte(identifier)): Gọi hàm **newGenesisBlock** để tạo khối genesis. Tham số truyền vào là một mảng byte được tạo từ chuỗi identifier. Có thể giả sử rằng identifier là một chuỗi đặc biệt để đại diện cho dữ liệu của khối genesis.

- err: Lưu trữ giá trị lỗi (nếu có) từ quá trình tạo khối genesis.

2. &**Blockchain**{ **Blocks**: []\*Block{genesisBlock}, }: Tạo một đối tượng blockchain mới. Các thuộc tính của blockchain bao gồm:

- **Blocks**: Là một slice chứa con trỏ đến các khối trong blockchain. Trong trường hợp này, chỉ có một khối duy nhất là genesisBlock.

3. return &**Blockchain**{ **Blocks**: []\*Block{genesisBlock}, }, err: Trả về con trỏ đến đối tượng blockchain với khối genesis đã được thêm vào, cũng như giá trị lỗi (nếu có) từ quá trình tạo khối genesis.

* Tóm lại, hàm **NewBlockchain** được sử dụng để tạo một đối tượng blockchain mới, chứa một khối genesis với dữ liệu được xác định bởi chuỗi identifier.
* Nếu tạo 1 genesis block với data rỗng, ta cần dùng hàm **generateRamdomByteArray()** để random data cho genesis block.
* Để thêm một block mới vào trong Chain ta sử dụng hàm func (bc \*Blockchain) **Add**(block \*Block) error. Với tham số truyền vào là một block và kết quả trả về là một lỗi nếu có. Trong hàm này gọi tới hàm ValidateBlockIntergrity() để xác thực một block cũng như PoW cho cả Blockchain.
* Sau khi đã tiến hành tạo và thêm một block vào trong chain Hàm **WriteToFile()** sẽ được sử dụng để lưu dữ liệu của chain vào trong file json.
* Trong trường hợp nếu có một file json chứa dữ liệu của một chain có sẵn thì ta sẽ gọi tới hàm **LoadChainData()** thông số đầu vào sẽ là đường dẫn tới file chứa dữ liệu đó. Bên trong thân hàm sẽ gọi tới hàm **ReadFromFile()** đọc dữ liệu vào trong biến “bc” rồi truyền vào **NewBlockChain()** để tạo.
* Để xem lại BlockChain mình đã tạo hoặc có sẵn thì sử dụng hàm **DisplayBlockData()** đã được trình bày bên trong trước trình. Hàm này sẽ hiện thị tất cả dữ liệu trong một block theo thứ từ đầu tới mới nhất.

1. **Merkle Tree**

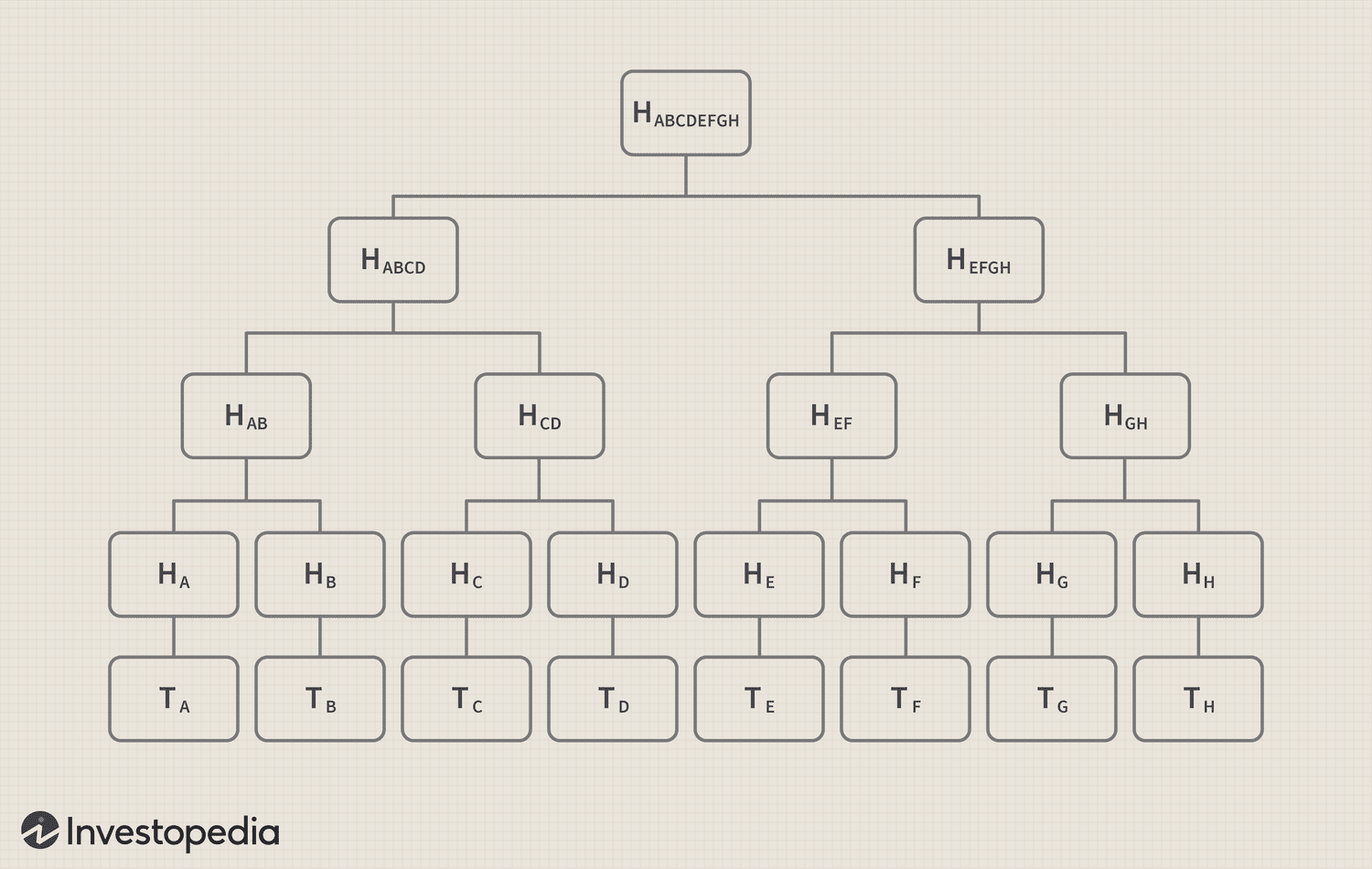
Cây Merkle, được khái niệm bởi Ralph Merkle, đóng vai trò quan trọng trong lĩnh vực khoa học máy tính và đã được áp dụng rộng rãi trong công nghệ blockchain. Cấu trúc cây này cung cấp một phương tiện an toàn và hiệu quả để tóm tắt và xác minh tính toàn vẹn của các tập dữ liệu lớn. Trong blockchain, chúng đóng vai trò quan trọng để tổng hợp và xác minh các giao dịch, đảm bảo tính đáng tin cậy và sức mạnh của tính toàn vẹn dữ liệu trong blockchain.

1. **Cấu trúc cây Merkle**

Cấu trúc của một Cây Merkle là sự kết hợp độc đáo giữa sự đơn giản và tính bảo mật mật mã. Nó bao gồm ba thành phần chính:

1. Nút Lá (Leaf Nodes): Mỗi nút lá trong Cây Merkle đại diện cho một khối dữ liệu cụ thể, có thể là một giao dịch đơn, một tài liệu, hoặc bất kỳ dữ liệu nào khác. Những nút này là các khối xây dựng cơ bản của cây và được tạo ra bằng cách hash dữ liệu mà chúng biểu thị, sử dụng một hàm băm mật mã như SHA-256, để đảm bảo tính toàn vẹn mật mã của dữ liệu.

2. Nút Không Lá (Non-Leaf Nodes): Những nút không phải lá, hay còn được gọi là nút trung gian, được tạo ra bằng cách băm sự kết hợp của các nút con của chúng. Quá trình này được áp dụng đệ quy, bắt đầu từ các nút lá và di chuyển lên trên, liên kết các nút theo cách mật mã. Hiệu ứng chuỗi này đảm bảo rằng bất kỳ sự thay đổi nào trong một phần dữ liệu cũng sẽ thay đổi giá trị băm của nút cha, giúp phát hiện được bất kỳ sự thay đổi không được ủy quyền nào.

3. Nút Gốc (Root Node): Ở đỉnh cây, gọi là nút gốc, là kết quả của quá trình băm từng bước. Hash được lưu trữ trong nút này, gọi là Merkle Root, là một hash duy nhất đại diện cho toàn bộ bộ dữ liệu ở phía dưới. Tính toàn vẹn của Merkle Root là quan trọng vì nó phản ánh tính toàn vẹn của toàn bộ cây Merkle.

1. **Xây dựng một cây Merkle trong Blockchain**

Trong chương trình, cây merkel được xây dưng bằng cách lấy tất cả dữ liệu transaction của một block sau đó hash đôi một với nhau. Cụ thể, func chính của việc xây dựng cây merkle là “buildTree”. Hướng xây dựng cây merkle là xây dựng từ thấp tới cao theo từng level. Nếu trường hợp số lượng node lẻ thì node cuối cùng tự nhân đôi nó lên để hash.

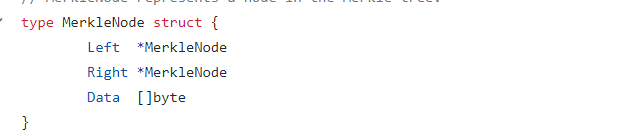
Và sử dụng SHA-256 để hash dữ liệu.

Về hoạt động của cây merkle. Nó là giá trị băm tóm tắt của tất cả các giao dịch trong một block và tạo dấu vân kỹ thuật số cho cả block. Cho phép người dùng xác minh đó có phải là giao dịch trong một khối hay không.

Dựa trên những gì đã được học tiến trình tạo cây merkle như sau:

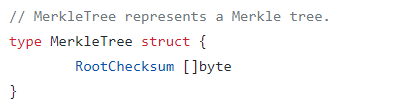
* + Tạo cấu trúc cho một node và một cây:

Cấu trúc của một MerkleNode.



Trong đó Left, Right lần lượt là các nút trái phải. Data là dữ liệu hash của một node.

Cấu trúc của một MekleTree.



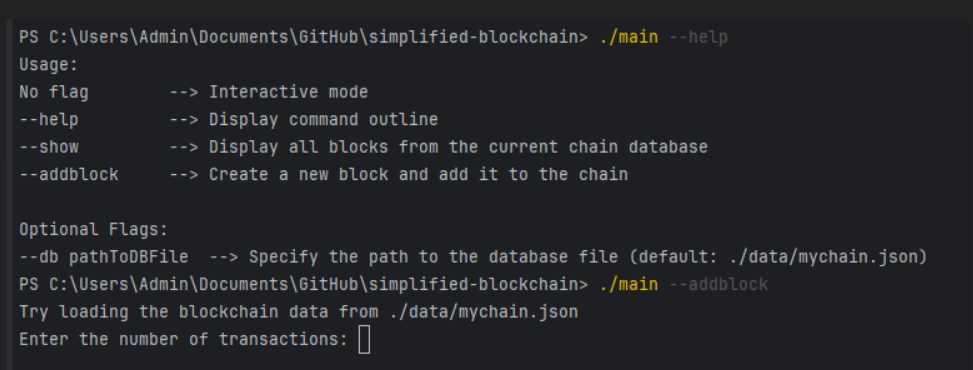
* + Cây mekle được tạo bằng cách băm các cặp nút liên tục cho đến khi chỉ còn lại một giá trị băm, trị băm này được gọi là Mekle Root trong chương trình được khai báo là RootCheckSum (Dùng để kiểm tra lại). Trong chương trình, việc này đã được giải quyết bằng cách sử dụng SHA-256 để hash dữ liệu theo cặp được viết trong hàm concatenateAddHash(...) bằng cách truyền 2 node trái phải vào và ghép lại rồi hash với nhau theo đúng định nghĩa của cây merkle.
  + Xây dựng cây từ dưới lên với các tham số truyền vào là dữ liệu của Transactions. Nếu số lượng Transactions đi vào là lẻ thì Transactions cuối cung tự thêm vào chính nó tức là giá trị hash của node trước nó(Đã được trình bày trong hàm buildTree()).
  + Về việc làm sao để xác thực thì đã được xây dựng ở hàm ValidateBlockIntergrity(). Bằng cách so sánh giá trị MerkleRoot.
  + Vì có tính xác thực như vậy nên Merkle Tree đã được ứng vào làm PoW cho blockchain. Trong chương trình đã sử dùng hàm ValidateBlockIntergrity() để kiểm tra trước khi add một block vào chuỗi.

Lợi ích của Merkle Tree trong Blockchain:

* + Xác thực tính toàn vẹn của dữ liệu.
  + Chiếm ít dung lượng đĩa.
  + Thông tin nhỏ khi trao đổi trên mạng
  + Xác minh hiệu quả.

1. **Command Line Client.**

Xây dựng một Menu CommanLine như sau :



Ta tạo các biến Flag như sau : helpFlag, showFlag, addBlockFlag. Thực hiện các chức năng sau.

* + HelpFlag được dùng khi ta gọi “-- help” sẽ hiển thị hướng dẫn dùng lệnh (hay command outline)
  + showFlag được dùng khi gọi “-- show” sẽ hiển thị dữ liệu của tất cả các block từ chain có trong Database theo biến dbPathFlag.
  + Biến dbPathFlag được dùng khi gọi “-- db(Được dùng thiết lập đường dẫn đến database đang lưu chain hiện có)” có 3 trường hợp sẽ xảy ra:
    - Nếu dùng -- db thì nó sẽ sử dụng đường dẫn ngay sau đó
    - Nếu không dùng – db thì sẽ kiểm tra trong MYCHAINPATH trong biến môi trường rồi lấy ra đường dẫn.
    - Trường hợp còn lại là đường dẫn mặc định ./data/mychain.json
  + addBlockFlag được dùng khi gọi “-- addblock” dùng để tạo block mới rồi thêm vào chain. Nó sẽ load chain theo đường dẫn của dbPath. Nếu không tồn tại đường dẫn đó thì báo lỗi. Ngược lại sẽ tạo ra block mới, sau đó add vào trong chain bằng hàm Add() đã được nói ở trên.
  + Nếu người dùng khi chạy không gõ “-- help” hoặc không có flag nào sử dụn thì chương trình tương tác được chạy