目录

一 任务

以《区块链 数据格式规范》为标准和参考，根据老师提供的原代码和文档，构建一条自己的原型区块链，必须实现交易记录和查询。不要求实现区块链网络功能。

二 设计过程

（完整代码详见附录）

（一）总体思路

参考《区块链 数据格式规范》和老师提供的文档，在所提供源代码的基础上进行增删修改，逐步完成区块、区块链的形成、工作量证明、数据库、序列化、持久化、命令行接口编写、地址模块编写、交易模块编写、UTXO 集、Merkle 树的实现等。利用命令行进行交互，实现区块链的各功能。

（二）详细过程

（1）区块 Block

区块链中真正存储有效信息的是区块（block）。区块包含了一些技术实现的相关信息，比如版本，当前时间戳和前一个区块的哈希。设计区块的数据结构如下：

```
type Block struct {
   Timestamp      int64
   Transactions   []*Transaction
   PrevBlockHash  []byte
   Hash           []byte
   Nonce          int
   Accounts       []*Account
```

```
    BDTypes        []*BlockDataType
    TDTypes        []*TransanctionDataType
    EDTypes        []*EntityDataType
    CDTypes        []*ContractDataType
    ADTypes        []*AssemblyDataType
}
```

从上至下依次是时间戳、交易记录、前一个块的哈希、本块的哈希、计数器、账户数据、区块数据、事务数据、实体数据、合约数据、配置数据。

其中账户数据、区块数据、事务数据、实体数据、合约数据和配置数据又被定义为结构体，如下：

```
type Account struct{
    AccountPublicKey   string //账户公钥
    AccountPrivateKey  string //账户私钥
    AccountAsset      []  int //账户资产
    DigitalCertificate[]  int //数字证书
    Institution       []  int //账户所属机构
}
```

账户数据包括账户公钥、账户私钥、账户资产、数字证书、账户所属机构这5项内容。

```
type BlockDataType struct{
    BlockHeight       int //区块高度
    BlockID          string //区块标识
    BlockVersion     string //版本信息
    MerkleTreeRoot string //默克尔树根
    Difficulty        int //难度系数
}
```

区块数据包括区块高度、区块标识、版本信息、默克尔根树、难度系数这5项内容。

```
type TransanctionDataType struct{
    TransactionID        string //事务标识
```

```
    TransactionType      string //事务类型
    Signers              string //签名者
    TransactionTimestamp string //事务时间戳
}
```

事务数据包括事务标识、事务类型、签名者、事务时间戳这 4 项内容。

```
type EntityDataType struct{
    SenderAddress    string //发起方地址
    RecipientAddress string //接收方地址
    TransactionAmount   int //事务处理发生额
    TransactionFee      int //事务处理费用
    AdditionalData   string //附加数据
    Memo             string //实体数据备注
    igners           string //签名者
    TransacTimestamp string //事务时间戳
}
```

实体数据包括发起方地址、接收方地址、事务处理发生额、事务处理发生费用、附加数据、实体数据备注、签名者、事务时间戳这 8 项内容。

```
type ContractDataType struct{
    ContrctID       string //合约标识
    ContractVersion string //合约版本号
    ContractCode    string //合约代码
    ContractStorage  []int //合约存储
}
```

合约数据包括合约标识、合约版本号、合约代码、合约存储这 4 项内容。

```
type AssemblyDataType struct{
    AssemblyVersion string //协议版本号
    SoftwareVersion string //软件版本号
    PeerID          string //节点标识
    PeerAddress     string //节点地址
```

```
    PeerPublicKey   string //节点公钥
}
```

配置数据包括协议版本号、软件版本号、节点标识、节点地址、节点公钥这5项内容。

在这里block.go中还有与块相关的多种功能，如添加块、序列化、逆序列化等等。

（2）区块链 Blockchain

尝试实现一个区块链原型：一个数组构成的一系列区块，每个块都与前一个块相关联。Blockchain结构体定义如下：

```
type Blockchain struct {
    tip []byte
    db  *bolt.DB
}
```

其中bolt.DB是go语言关于数据库操作的一个包中定义的数据类型。

在这里blockchain.go中还有与区块链相关的多种功能，如创建区块链、对交易签名、证实交易等等。

（3）工作量证明 Proof of Work

构造 ProofOfWork结构体如下：

```
type ProofOfWork struct {
    block  *Block
    target *big.Int
}
```

里面存储了指向一个块(block)和一个目标(target)的指针。这里的“目标”，也就是挖矿成功的必要条件。这里使用了一个大整数，将哈希与目标进行比较：先把哈希转换成一个大整数，然后检测它是否小于目标。

在NewProofOfWork函数中，可以将big.Int初始化为1，然后左移256 - targetBits位。256是一个SHA-256哈希的位数，将要使用的是SHA-256哈希算

法。

（4）序列化与持久化 Serialization and Persistence

本质上，区块链是一个分布式数据库，如何存储是一个问题。

如果不使用数据库，而是在每次运行程序时，简单地将区块链存储在内存中。那么一旦程序退出，所有的内容就都消失了。这样以来就没有办法再次使用这条链，也没有办法与其他人共享，所以需要把它存储到磁盘上。

由于 BoltDB 非常简洁，用 Go 实现，不需要运行服务器，能够允许构造想要的数据结构，可以选择 BoltDB 为使用的数据库。不过数据库中存储的是键值对，并且使用字节数组，所以涉及到编码解码的工作，也即序列化，使用 encoding/god 来实现的方法如下：

```
func (b *Block) Serialize() []byte {
   var result bytes.Buffer
   encoder := gob.NewEncoder(&result)

   err := encoder.Encode(b)
   if err != nil {
      log.Panic(err)
   }

   return result.Bytes()
}

func DeserializeBlock(d []byte) *Block {
   var block Block

   decoder := gob.NewDecoder(bytes.NewReader(d))
   err := decoder.Decode(&block)
   if err != nil {
      log.Panic(err)
```

```
    }

    return &block
}
```

而关于持久化，我们希望 NewBlockchain 能够完成：

1. 打开一个数据库文件
2. 检查文件里面是否已经存储了一个区块链
3. 如果已经存储了一个区块链：
    3.1. 创建一个新的 Blockchain 实例
    3.2. 设置 Blockchain 实例的 tip 为数据库中存储的最后一个块的哈希
4. 如果没有区块链：
    4.1. 创建创世块
    4.2. 存储到数据库
    4.3. 将创世块哈希保存为最后一个块的哈希
    4.4. 创建一个新的 Blockchain 实例，初始时 tip 指向创世块

实现代码如下：

```
func NewBlockchain() *Blockchain {
    if dbExists() == false {
        fmt.Println("No existing blockchain found. Create one first.")
        os.Exit(1)
    }

    var tip []byte
    db, err := bolt.Open(dbFile, 0600, nil)
    if err != nil {
        log.Panic(err)
    }

    err = db.Update(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(blocksBucket))
        tip = b.Get([]byte("l"))
        return nil
```

```
    })
    if err != nil {
        log.Panic(err)
    }
    bc := Blockchain{tip, db}
    return &bc
}
```

（5）命令行界面 Command-line Interface

为了优化用户体验，希望提供一个与程序交互的接口：不能只是在 main 函数中简单执行 NewBlockchain 和 bc.AddBlock，而应该在命令行输入命令。所有命令行相关的操作都会通过 CLI 结构进行处理，将 Run 函数作为入口，使用标准库里面的 flag 包来解析命令行参数，检查用户提供的命令，解析相关的 flag 子命令，接着检查解析是哪一个子命令，并调用相关函数，并使用 BlockchainIterator 对区块链中的区块进行迭代。

部分实现代码如下：

```
func (cli *CLI) Run() {
    cli.validateArgs()

    getBalanceCmd := flag.NewFlagSet("getbalance", flag.ExitOnError)
    createBlockchainCmd := flag.NewFlagSet("createblockchain", flag.ExitOnError)
    createWalletCmd := flag.NewFlagSet("createwallet", flag.ExitOnError)
    listAddressesCmd := flag.NewFlagSet("listaddresses", flag.ExitOnError)
    printChainCmd := flag.NewFlagSet("printchain", flag.ExitOnError)
    reindexUTXOCmd := flag.NewFlagSet("reindexutxo", flag.ExitOnError)
    sendCmd := flag.NewFlagSet("send", flag.ExitOnError)

    getBalanceAddress := getBalanceCmd.String("address", "", "The address to get
balance for")
    createBlockchainAddress := createBlockchainCmd.String("address", "", "The
```

```
address to send genesis block reward to")
   sendFrom := sendCmd.String("from", "", "Source wallet address")
   sendTo := sendCmd.String("to", "", "Destination wallet address")
   sendAmount := sendCmd.Int("amount", 0, "Amount to send")

   switch os.Args[1] {
   case "getbalance":
      err := getBalanceCmd.Parse(os.Args[2:])
      if err != nil {
         log.Panic(err)
      }
   case "createblockchain":
      err := createBlockchainCmd.Parse(os.Args[2:])
      if err != nil {
         log.Panic(err)
      }
   case "createwallet":
      err := createWalletCmd.Parse(os.Args[2:])
      if err != nil {
         log.Panic(err)
      }
   case "listaddresses":
      err := listAddressesCmd.Parse(os.Args[2:])
      if err != nil {
         log.Panic(err)
      }
   case "printchain":
      err := printChainCmd.Parse(os.Args[2:])
      if err != nil {
         log.Panic(err)
      }
   case "send":
```

```
        err := sendCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    case "reindexutxo":
        err := reindexUTXOCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    default:
        cli.printUsage()
        os.Exit(1)
    }

    if getBalanceCmd.Parsed() {
        if *getBalanceAddress == "" {
            getBalanceCmd.Usage()
            os.Exit(1)
        }
        cli.getBalance(*getBalanceAddress)
    }

    if createBlockchainCmd.Parsed() {
        if *createBlockchainAddress == "" {
            createBlockchainCmd.Usage()
            os.Exit(1)
        }
        cli.createBlockchain(*createBlockchainAddress)
    }

    if createWalletCmd.Parsed() {
        cli.createWallet()
```

```
    }

    if listAddressesCmd.Parsed() {
        cli.listAddresses()
    }

    if printChainCmd.Parsed() {
        cli.printChain()
    }

    if reindexUTXOCmd.Parsed() {
        cli.reindexUTXO()
    }

    if sendCmd.Parsed() {
        if *sendFrom == "" || *sendTo == "" || *sendAmount <= 0 {
            sendCmd.Usage()
            os.Exit(1)
        }

        cli.send(*sendFrom, *sendTo, *sendAmount)
    }
}
```

（6）地址 Address

为了得到更加真实的效果，我们也希望生成真实的用户地址。

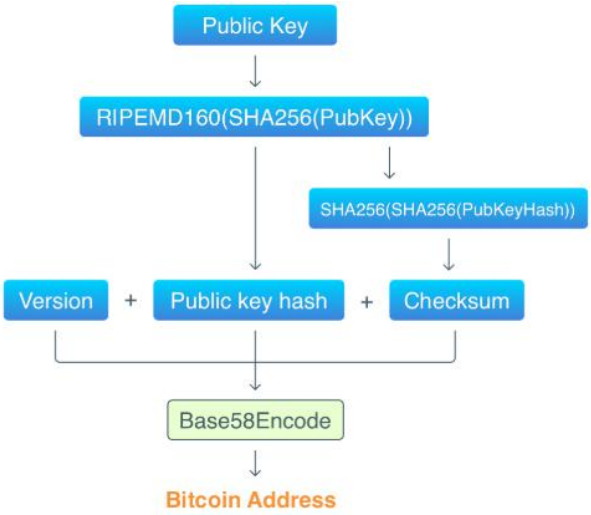
使用如下步骤可以将一个公钥转换成 Base58 地址：

1. 使用 RIPEMD160(SHA256(PubKey))哈希算法，取公钥并对其哈希两次
2. 给哈希加上地址生成算法版本的前缀
3. 对于第二步生成的结果，使用 SHA256(SHA256(payload))再哈希，计算校

验和。校验和是结果哈希的前四个字节。

4. 将校验和附加到 version+PubKeyHash 的组合中。
5. 使用 Base58 对 version+PubKeyHash+checksum 组合进行编码。

下图为过程的示意图：



部分实现代码如下：

```
type Wallet struct {
   PrivateKey ecdsa.PrivateKey
   PublicKey  []byte
}

type Wallets struct {
   Wallets map[string]*Wallet
}

func NewWallets() (*Wallets, error) {
   wallets := Wallets{}
   wallets.Wallets = make(map[string]*Wallet)

   err := wallets.LoadFromFile()

func newKeyPair() (ecdsa.PrivateKey, []byte) {
   curve := elliptic.P256()
```

```
    private, err := ecdsa.GenerateKey(curve, rand.Reader)
    if err != nil {
        log.Panic(err)
    }
    pubKey := append(private.PublicKey.X.Bytes(), private.PublicKey.Y.Bytes()...)

    return *private, pubKey
}

    return &wallets, err
}
```

（7）交易 Transactions

该部分代码过多，详见附录。

UTXO 模型

还没有被下一个交易花费的 Output 被称为 UTXO：Unspent Transaction Output，即未花费交易输出。给定任何一个区块，计算当前所有的 UXTO 金额之和，等同于自创世区块到给定区块的挖矿奖励之和。

在钱包程序中，钱包管理的是一组私钥，对应的是一组公钥和地址。钱包程序必须从创世区块开始扫描每一笔交易，如果遇到某笔交易的某个 Output 是钱包管理的地址之一，则钱包余额增加；遇到某笔交易的某个 Input 是钱包管理的地址之一，则钱包余额减少。钱包的当前余额总是钱包地址关联的所有 UTXO 金额之和。

每一笔比特币交易都会创造输出，输出都会被区块链记录下来。给某个人发送比特币，实际上意味着创造新的 UTXO 并注册到那个人的地址，可以为他所用。

奖励

挖矿奖励（一笔 coinbase 交易）：当一个挖矿节点开始挖出一个新块时，它会将交易从队列中取出，并在前面附加一笔 coinbase 交易。coinbase 交易只有一

个输出，里面包含了矿工的公钥哈希。

UTXO 集

设计一个 UTXO 集，它是一个巨大的缓存，在所有区块链交易中构建索引而得到（对区块进行迭代，但是只须做一次），然后用它来计算余额和验证新的交易。

可以考虑使用以下方法：

1. Blockchain.FindUTXO - 通过对区块进行迭代找到所有未花费输出。

2. UTXOSet.Reindex - 使用 UTXO 找到未花费输出，然后在数据库中进行存储。这里就是缓存的地方。

3.UTXOSet.FindSpendableOutputs - 类似 Blockchain.FindSpendableOutputs，但是使用 UTXO 集。

4. UTXOSet.FindUTXO - 类似 Blockchain.FindUTXO，但是使用 UTXO 集。

5. Blockchain.FindTransaction 则无变动。

使用一个单一数据库，但是将 UTXO 集从存储在不同的 bucket 中。

这样以来，交易就已经被分开存储：实际交易被存储在区块链中，未花费输出被存储在 UTXO 集中。

Merkle 树

每个块都会有一个 Merkle 树，它从叶子节点（树的底部）开始，一个叶子节点就是一个交易哈希（比特币使用双 SHA256 哈希）。

从下往上，两两成对，连接两个节点哈希，将组合哈希作为新的哈希。新的哈希就成为新的树节点。重复该过程，直到仅有一个节点，也就是树根。根哈希然后就会当做是整个块交易的唯一标示，将它保存到区块头，然后用于工作量证明。

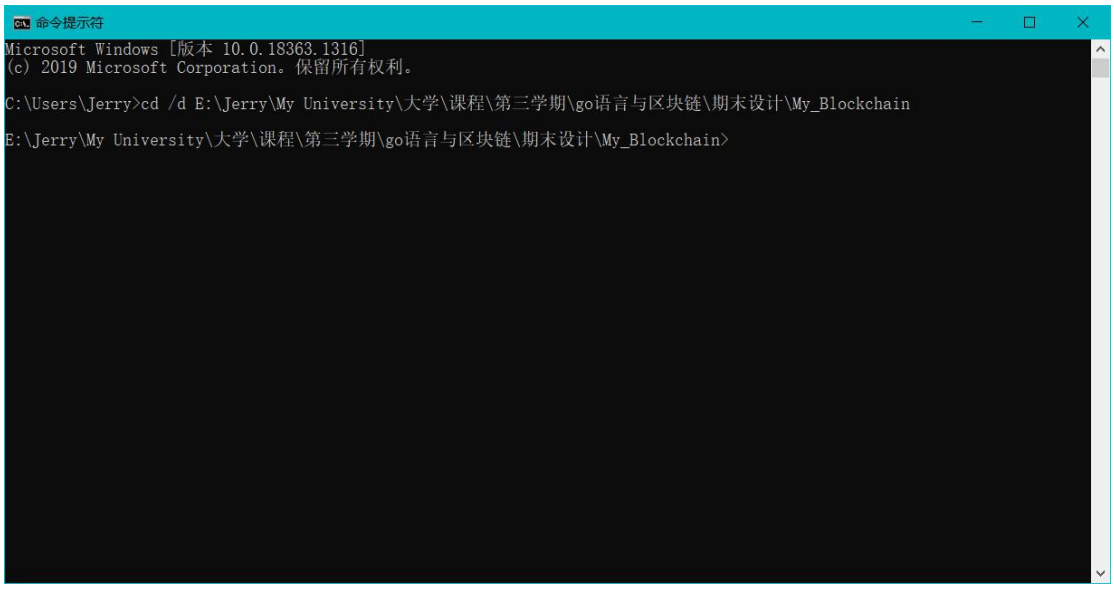
Merkle 树的好处就是一个节点可以在不下载整个块的情况下，验证是否包含某笔交易。并且这些只需要一个交易哈希，一个 Merkle 树根哈希和一个 Merkle 路径。

（8）网络 Network

对该部分我做了一些有益的尝试，不过最终未能跑通得到自己想要的结果，看来自己对 TCP/IP 协议的掌握程度还是不足，未能完成对去中心化的实现。同时该部分不是期末设计的核心要求，故不再叙述。

三 功能测试

首先打开命令行，利用 cd /d 加上 go 程序所在文件夹，前进到该路径下准备运行 go 程序。



使用命令 go run ./ wallet 可以生成地址，操作几次，我们可以得到多个地址，并进行之后的操作。

go run 加上./可以同时运行该文件夹下的所有 go 程序，避免了运行单一程序造成找不到文件而报错。而“./”之后的内容为我们输入的命令，通过 CLI 解析来完成相应功能。

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>
```

使用 listaddresses 命令可以获得当前生成的所有地址：

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ listaddresses
1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
1FBWEkmfUr6kvuMM2gZJu9fHsolPgFhQn3
1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
1FXDQMRLXhFZeuntZKY9yDQ9uyUNbFgp6i
1MMsuA1THnCmgSnrNiBKXyCaBwjuf7Uzsu
1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
164pp1PUQ3ykrictxQrte9RJ5eHEf27Fdt
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
```

选取其中的四个地址进行操作，给每个地址一个代号方便称呼。做成表格更加清楚：

代号	地址
Jerry	1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Paul	1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Henry	1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
James	1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve

下面 Jerry 开始挖矿，使用已经实现的 createblockchain 命令，将地址设置为 Jerry（1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL）。如图：

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createblockchain -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6

Done!
```

经过一段时间（约 15 分钟）的挖矿，成功挖到创世块，Jerry 获得 10BTC。

用 getbalance 查询各地址的余额如下：

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 10

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 0

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 0

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0
```

可以看到初始时除了 Jerry（1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL）的其他各地址的钱包余额都是 0。

代号	地址	余额
Jerry	1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL	10
Paul	1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa	0
Henry	1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7	0
James	1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve	0

下面我们测试各节点间的交易，使用 send 命令，给出发送方(from)、接收方(to)和金额(amount)就能完成，经过一段时间的挖矿就可以得到一个新块，并完成交易。

设定如下的交易模式：

首先由 Jerry 向 Paul 转 8BTC，向 Henry 转 1BTC：

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7P
usPmWvaVkfrAdSL -to 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa -amount 8
000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4

Success!

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7P
usPmWvaVkfrAdSL -to 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7 -amount 1
000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b

Success!
```

查询，得到更新后的余额如下：

代号	地址	余额
Jerry	1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL	1
Paul	1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa	8
Henry	1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7	1
James	1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve	0

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1Eab
Qnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 1

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBox
v6TcUjpCJTnGQxSgf72NHZa                    n
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 8

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvzt
t2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 1

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z
9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0
```

运行时间很长，挖矿时间最长的一块已经超过 1 个小时。原因是难度设置较高。

然后 Paul 向 Henry 转 7BTC。

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1AGuGAjBoSxv6TcUjpC
JTnGQxSgf72NHZa -to 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7 -amount 7
000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186

Success!
```

查询，得到更新后的余额如下：

代号	地址	余额
Jerry	1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL	1
Paul	1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa	1
Henry	1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7	8
James	1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve	0

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1Eab
Qnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 1

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoS
xv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 1

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvzt
t2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 8

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z
9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0
```

最后 Jerry、Paul、Henry 分别向 James 转 1、1、8TC：

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7P
usPmWvaVkfrAdSL -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 1
00000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d

Success!

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1AGuGAjBoSxv6TcUjpC
JTnGQxSgf72NHZa -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 1
000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033

Success!

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1F2uQPKvztt2bPji9Jb
8tfBNeP1m46zSu7 -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 8
00000065fab8754aa3938724297c70cadb95e92fdccc95fc42371cc8622825d5

Success!
```

查询，得到更新后的余额如下：

代号	地址	余额
Jerry	1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL	0
Paul	1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa	0
Henry	1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7	0
James	1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve	10

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 0

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 0

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7': 0

E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 10
```

下面使用 printchain 命令打印整个区块链，效果如下图所示，可以看到所有的交易记录都在其中。

```
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ printchain
============ Block 00000065fab8754aa3938724297c70cadb95e92fdccc95fc42371cc8622825d5 ============
Prev. block: 000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033
PoW: true

--- Transaction 175f19bec5fc610796bb8d88d61e128af591aec4ff2c5e84ebf9632cbda01c7a:
    Input 0:
      TXID:      9587e41c06ddfac3a5623138f42bc34dca1bf5efe8eb71c2ebd977f51e78d794
      Out:       0
      Signature: bf5b80abe9ffd07ab9d7e52f8b95e28830443ae7e6295c459913e63a5bcf5e3248e9173dc5cce5fdb9c3a09e58bf3549b9b0eb9116b43edc20c1fe365b1da361
      PubKey:    daf35edd085eb050e2961bd547f17acb5f016602a0a3c86f3c78fd0c1b20dc1310c8269baf10b322383cba744424ea7089e13f048bac37ab38d2ac2798f82751
    Input 1:
      TXID:      107cdc7f1840f116c7d292970215e819ddc06333b1c55c26464f99ea758b357d
      Out:       0
      Signature: 607ad77f592deb2b5b802d7679f9283c717cab2d3df09b95ed6bbe24162ee2a1dac41832f481aa04245b94d307eecf0ea440ffaa29d3695c71af48e3663faac9
      PubKey:    daf35edd085eb050e2961bd547f17acb5f016602a0a3c86f3c78fd0c1b20dc1310c8269baf10b322383cba744424ea7089e13f048bac37ab38d2ac2798f82751
    Output 0:
      Value:  8
      Script: fb352694109e8ae2dd47e1189e4bcb0e5fcac8cf


============ Block 000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033 ============
Prev. block: 00000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d
PoW: true

--- Transaction 0780efe0263f6fb0a1ac0b9137425c5ba5e24337f2ae74aa55447fdd5cf885a5:
    Input 0:
      TXID:      9587e41c06ddfac3a5623138f42bc34dca1bf5efe8eb71c2ebd977f51e78d794
      Out:       1
      Signature: 2edc61700f0e00874111011fd80cd08e6999805e447e7cacb664bd34b14fdaf941ef1c88e62b0c8b4d35eab4d3af600b8d6a28d3b5a1f61d50f2ad7e12de5da4
      PubKey:    4a26b8be612c2d8b84397f176168628b2392d16b8683c05b9198c7c194ace61f6af9fb85da432c67dcea45175cb72c2ef44e8eda3f04d4fa6b08ab00d1e1ea94
    Output 0:
      Value:  1
      Script: fb352694109e8ae2dd47e1189e4bcb0e5fcac8cf


============ Block 00000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d ============
Prev. block: 000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186
PoW: true

--- Transaction dbb7a7fdd1ba3be6a2602199ba9abc12ba6953df9c479d2d91d90631e3532c4a:
    Input 0:
      TXID:      107cdc7f1840f116c7d292970215e819ddc06333b1c55c26464f99ea758b357d
      Out:       1
      Signature: ba17f1648a999075f2b0d4e3b8aed83623e95849eefb6ad2c77dc64b879d3da3c53a3dd6fb1e51827739c29fd6bdb056616d0dc465cb4c79af142c1a65f685c0
      PubKey:    a7e85e0623eb5ba93a77246705db6fcd15b571f21ca5839e7920ff5bb69a3200889771143f2c4f971d55b7c559294647f6fbb656ad54ab4974a04499cb40af30
    Output 0:
      Value:  1
      Script: fb352694109e8ae2dd47e1189e4bcb0e5fcac8cf


============ Block 000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186 ============
Prev. block: 000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b
PoW: true

--- Transaction 9587e41c06ddfac3a5623138f42bc34dca1bf5efe8eb71c2ebd977f51e78d794:
    Input 0:
      TXID:      918b1c5bc438d6a042811bd08effb61afb983fd80c039a3ff37faa3af05ce45f
      Out:       0
      Signature: 6c34a4c293f280f64e4a6113fabb2ab9dbb0bd7befdfa6831e5205f22f505c1f76ca449f3389d829970c05f7170a054e2df60651541a4c5bab5dd7dabd444a16
      PubKey:    4a26b8be612c2d8b84397f176168628b2392d16b8683c05b9198c7c194ace61f6af9fb85da432c67dcea45175cb72c2ef44e8eda3f04d4fa6b08ab00d1e1ea94
    Output 0:
      Value:  7
      Script: 99edebc0fdf519b5d863fa7f92e72c1e530f1736
    Output 1:
      Value:  1
      Script: 65baff48208d3accdbcabe47f1ae76b996275d07


============ Block 000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b ============
Prev. block: 000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4
PoW: true

--- Transaction 107cdc7f1840f116c7d292970215e819ddc06333b1c55c26464f99ea758b357d:
    Input 0:
      TXID:      918b1c5bc438d6a042811bd08effb61afb983fd80c039a3ff37faa3af05ce45f
      Out:       1
      Signature: b65e32f50f94f438a9e8e9468099f1dc3fd71a1853ee2381e6a24421c8992d158a989341d03364ded19c1a449cdc8c53c87a0f1871c8950736082b82fef95f3c
      PubKey:    a7e85e0623eb5ba93a77246705db6fcd15b571f21ca5839e7920ff5bb69a3200889771143f2c4f971d55b7c559294647f6fbb656ad54ab4974a04499cb40af30
    Output 0:
      Value:  1
      Script: 99edebc0fdf519b5d863fa7f92e72c1e530f1736
    Output 1:
      Value:  1
      Script: bd6433d7a4f3e01b2416dcc7828f63064511b697


============ Block 000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4 ============
Prev. block: 000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6
PoW: true

--- Transaction 918b1c5bc438d6a042811bd08effb61afb983fd80c039a3ff37faa3af05ce45f:
    Input 0:
      TXID:      565dc3b93c0e62689475e13b957453009e2289b02d94a32d0d75ea144d8dd07f
      Out:       0
      Signature: 3d191788c43d0ba2a2025691c1f6f57caba4448ad7b8c921429728ee86e5f0ba8356e781857116865290dfcd03d982a8a68c0614899e7003d1c0347aa483e2df
      PubKey:    a7e85e0623eb5ba93a77246705db6fcd15b571f21ca5839e7920ff5bb69a3200889771143f2c4f971d55b7c559294647f6fbb656ad54ab4974a04499cb40af30
    Output 0:
      Value:  8
      Script: 65baff48208d3accdbcabe47f1ae76b996275d07
    Output 1:
      Value:  2
      Script: bd6433d7a4f3e01b2416dcc7828f63064511b697


============ Block 000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6 ============
Prev. block:
PoW: true

--- Transaction 565dc3b93c0e62689475e13b957453009e2289b02d94a32d0d75ea144d8dd07f:
    Input 0:
      TXID:
      Out:       -1
      Signature:
      PubKey:    54686520546696d65732030332f4a616e2f323030392043686616e63656c6c6f72206f6e206272696e6b206f662073656365726e64206261696c6f757420666f7220626
    Output 0:
      Value:  10
      Script: bd6433d7a4f3e01b2416dcc7828f63064511b697
```

在以上叙述过程中已经完成所有交易记录和查询的内容，很多其他功能也在其中。至此，我们已经完成所有实现功能的测试！

四 总结

在本次项目中成功搭建了一条自己的原型区块链，完成了区块、区块链的数据结构设计、工作量证明机制设计、区块链的序列化和持久化、命令行界面交互、区块链用户的地址设计、交易记录、查询等多个内容。并且针对期末设计的任务做了一些测试，通过了要求，同时额外完成了一些诸如美化界面、增加程序鲁棒性、区块链网络部分初探的内容。

通过亲手实践，我对区块链技术的诸多细节更加了解，更体会到了用 go 语言来处理区块链的便捷之处。该项目是第三学期的最后一个项目，项目的完成也为第三学期画上一个圆满的句号。希望在以后的课程中能够使用到 go 语言，并且能够继续运用区块链的思想。

附录

主程序

main.go

```
package main

func main() {
   cli := CLI{}
   cli.Run()
}
```

区块链相关结构

block.go

```
package main

import (
   "bytes"
   "encoding/gob"
   "log"
   "time"
)

// Block represents a block in the blockchain
type Block struct {
   Timestamp     int64
   Transactions   []*Transaction
   PrevBlockHash  []byte
   Hash          []byte
   Nonce         int
   Accounts      []*Account
   BDTypes       []*BlockDataType
   TDTypes       []*TransanctionDataType
   EDTypes       []*EntityDataType
   CDTypes       []*ContractDataType
   ADTypes       []*AssemblyDataType
}

type Account struct{
   AccountPublicKey   string //账户公钥
   AccountPrivateKey  string //账户私钥
   AccountAsset     []  int //账户资产
```

```
    DigitalCertificate[]  int //数字证书
    Institution       []  int //账户所属机构
}

type BlockDataType struct{
    BlockHeight       int //区块高度
    BlockID         string //区块标识
    BlockVersion    string //版本信息
    MerkleTreeRoot string //默克尔树根
    Difficulty        int //难度系数
}

type TransanctionDataType struct{
    TransactionID        string //事务标识
    TransactionType      string //事务类型
    Signers              string //签名者
    TransactionTimestamp string //事务时间戳
}

type EntityDataType struct{
    SenderAddress    string //发起方地址
    RecipientAddress string //接收方地址
    TransactionAmount   int //事务处理发生额
    TransactionFee      int //事务处理费用
    AdditionalData   string //附加数据
    Memo             string //实体数据备注
    igners           string //签名者
    TransacTimestamp string //事务时间戳
}

type ContractDataType struct{
    ContrctID       string //合约标识
```

```
    ContractVersion string //合约版本号
    ContractCode    string //合约代码
    ContractStorage  []int //合约存储
}

type AssemblyDataType struct{
    AssemblyVersion string //协议版本号
    SoftwareVersion string //软件版本号
    PeerID          string //节点标识
    PeerAddress     string //节点地址
    PeerPublicKey   string //节点公钥
}

// NewBlock creates and returns Block
func NewBlock(transactions []*Transaction, prevBlockHash []byte) *Block {
    block := &Block{time.Now().Unix(), transactions, prevBlockHash, []byte{}, 0}
    pow := NewProofOfWork(block)
    nonce, hash := pow.Run()
    block.Hash = hash[:]
    block.Nonce = nonce
    return block
}

// NewGenesisBlock creates and returns genesis Block
func NewGenesisBlock(coinbase *Transaction) *Block {
    return NewBlock([]*Transaction{coinbase}, []byte{})
}

// HashTransactions returns a hash of the transactions in the block
func (b *Block) HashTransactions() []byte {
    var transactions [][]byte
    for _, tx := range b.Transactions {
```

```
        transactions = append(transactions, tx.Serialize())
    }
    mTree := NewMerkleTree(transactions)

    return mTree.RootNode.Data
}

// Serialize serializes the block
func (b *Block) Serialize() []byte {
    var result bytes.Buffer
    encoder := gob.NewEncoder(&result)
    err := encoder.Encode(b)
    if err != nil {
        log.Panic(err)
    }
    return result.Bytes()
}

// DeserializeBlock deserializes a block
func DeserializeBlock(d []byte) *Block {
    var block Block
    decoder := gob.NewDecoder(bytes.NewReader(d))
    err := decoder.Decode(&block)
    if err != nil {
        log.Panic(err)
    }
    return &block
}
```

blockchain.go

```
package main

import (
   "bytes"
   "crypto/ecdsa"
   "encoding/hex"
   "errors"
   "fmt"
   "log"
   "os"

   "github.com/boltdb/bolt"
)

const dbFile = "blockchain.db"
const blocksBucket = "blocks"
const genesisCoinbaseData = "The Times 03/Jan/2009 Chancellor on brink of second
bailout for banks"

// Blockchain implements interactions with a DB
type Blockchain struct {
   tip []byte
   db  *bolt.DB
}

// CreateBlockchain creates a new blockchain DB
func CreateBlockchain(address string) *Blockchain {
   if dbExists() {
      fmt.Println("Blockchain already exists.")
      os.Exit(1)
```

```
}

var tip []byte

cbtx := NewCoinbaseTX(address, genesisCoinbaseData)
genesis := NewGenesisBlock(cbtx)
db, err := bolt.Open(dbFile, 0600, nil)
if err != nil {
   log.Panic(err)
}

err = db.Update(func(tx *bolt.Tx) error {
   b, err := tx.CreateBucket([]byte(blocksBucket))
   if err != nil {
      log.Panic(err)
   }
   err = b.Put(genesis.Hash, genesis.Serialize())
   if err != nil {
      log.Panic(err)
   }
   err = b.Put([]byte("l"), genesis.Hash)
   if err != nil {
      log.Panic(err)
   }
   tip = genesis.Hash
   return nil
})
if err != nil {
   log.Panic(err)
}
bc := Blockchain{tip, db}
return &bc
```

```
}

// NewBlockchain creates a new Blockchain with genesis Block
func NewBlockchain() *Blockchain {
   if dbExists() == false {
      fmt.Println("No existing blockchain found. Create one first.")
      os.Exit(1)
   }

   var tip []byte
   db, err := bolt.Open(dbFile, 0600, nil)
   if err != nil {
      log.Panic(err)
   }

   err = db.Update(func(tx *bolt.Tx) error {
      b := tx.Bucket([]byte(blocksBucket))
      tip = b.Get([]byte("l"))
      return nil
   })
   if err != nil {
      log.Panic(err)
   }
   bc := Blockchain{tip, db}
   return &bc
}

// FindTransaction finds a transaction by its ID
func (bc *Blockchain) FindTransaction(ID []byte) (Transaction, error) {
   bci := bc.Iterator()
   for {
      block := bci.Next()
```

```
            for _, tx := range block.Transactions {
                if bytes.Compare(tx.ID, ID) == 0 {
                    return *tx, nil
                }
            }
            if len(block.PrevBlockHash) == 0 {
                break
            }
        }

        return Transaction{}, errors.New("Transaction is not found")
    }

    // FindUTXO finds all unspent transaction outputs and returns transactions with spent
    outputs removed
    func (bc *Blockchain) FindUTXO() map[string]TXOutputs {
        UTXO := make(map[string]TXOutputs)
        spentTXOs := make(map[string][]int)
        bci := bc.Iterator()

        for {
            block := bci.Next()

            for _, tx := range block.Transactions {
                txID := hex.EncodeToString(tx.ID)

            Outputs:
                for outIdx, out := range tx.Vout {
                    // Was the output spent?
                    if spentTXOs[txID] != nil {
                        for _, spentOutIdx := range spentTXOs[txID] {
```

```
                if spentOutIdx == outIdx {
                    continue Outputs
                }
            }
        }

        outs := UTXO[txID]
        outs.Outputs = append(outs.Outputs, out)
        UTXO[txID] = outs
      }

      if tx.IsCoinbase() == false {
        for _, in := range tx.Vin {
          inTxID := hex.EncodeToString(in.Txid)
          spentTXOs[inTxID] = append(spentTXOs[inTxID], in.Vout)
        }
      }
    }

    if len(block.PrevBlockHash) == 0 {
      break
    }
  }

  return UTXO
}

// Iterator returns a BlockchainIterat
func (bc *Blockchain) Iterator() *BlockchainIterator {
  bci := &BlockchainIterator{bc.tip, bc.db}

  return bci
```

```
}

// MineBlock mines a new block with the provided transactions
func (bc *Blockchain) MineBlock(transactions []*Transaction) *Block {
   var lastHash []byte

   for _, tx := range transactions {
      if bc.VerifyTransaction(tx) != true {
         log.Panic("ERROR: Invalid transaction")
      }
   }

   err := bc.db.View(func(tx *bolt.Tx) error {
      b := tx.Bucket([]byte(blocksBucket))
      lastHash = b.Get([]byte("l"))

      return nil
   })
   if err != nil {
      log.Panic(err)
   }

   newBlock := NewBlock(transactions, lastHash)

   err = bc.db.Update(func(tx *bolt.Tx) error {
      b := tx.Bucket([]byte(blocksBucket))
      err := b.Put(newBlock.Hash, newBlock.Serialize())
      if err != nil {
         log.Panic(err)
      }

      err = b.Put([]byte("l"), newBlock.Hash)
```

```
            if err != nil {
                log.Panic(err)
            }

            bc.tip = newBlock.Hash

            return nil
        })
        if err != nil {
            log.Panic(err)
        }

        return newBlock
    }

    // SignTransaction signs inputs of a Transaction
    func (bc *Blockchain) SignTransaction(tx *Transaction, privKey ecdsa.PrivateKey)
    {
        prevTXs := make(map[string]Transaction)

        for _, vin := range tx.Vin {
            prevTX, err := bc.FindTransaction(vin.Txid)
            if err != nil {
                log.Panic(err)
            }
            prevTXs[hex.EncodeToString(prevTX.ID)] = prevTX
        }
        tx.Sign(privKey, prevTXs)
    }

    // VerifyTransaction verifies transaction input signatures
    func (bc *Blockchain) VerifyTransaction(tx *Transaction) bool {
```

```
    if tx.IsCoinbase() {
        return true
    }

    prevTXs := make(map[string]Transaction)
    for _, vin := range tx.Vin {
        prevTX, err := bc.FindTransaction(vin.Txid)
        if err != nil {
            log.Panic(err)
        }
        prevTXs[hex.EncodeToString(prevTX.ID)] = prevTX
    }

    return tx.Verify(prevTXs)
}

func dbExists() bool {
    if _, err := os.Stat(dbFile); os.IsNotExist(err) {
        return false
    }
    return true
}
```

blockchain_iterator

```
package main

import (
    "log"

    "github.com/boltdb/bolt"
```

```
)

// BlockchainIterator is used to iterate over blockchain blocks
type BlockchainIterator struct {
   currentHash []byte
   db          *bolt.DB
}

// Next returns next block starting from the tip
func (i *BlockchainIterator) Next() *Block {
   var block *Block

   err := i.db.View(func(tx *bolt.Tx) error {
      b := tx.Bucket([]byte(blocksBucket))
      encodedBlock := b.Get(i.currentHash)
      block = DeserializeBlock(encodedBlock)

      return nil
   })

   if err != nil {
      log.Panic(err)
   }

   i.currentHash = block.PrevBlockHash

   return block
}
```

默克尔树相关文件

merkle_tree.go

```
package main

import (
   "crypto/sha256"
)

// MerkleTree represent a Merkle tree
type MerkleTree struct {
   RootNode *MerkleNode
}

// MerkleNode represent a Merkle tree node
type MerkleNode struct {
   Left  *MerkleNode
   Right *MerkleNode
   Data  []byte
}

// NewMerkleTree creates a new Merkle tree from a sequence of data
func NewMerkleTree(data [][]byte) *MerkleTree {
   var nodes []MerkleNode

   if len(data)%2 != 0 {
      data = append(data, data[len(data)-1])
   }

   for _, datum := range data {
      node := NewMerkleNode(nil, nil, datum)
```

```
        nodes = append(nodes, *node)
    }

    for i := 0; i < len(data)/2; i++ {
        var newLevel []MerkleNode

        for j := 0; j < len(nodes); j += 2 {
            node := NewMerkleNode(&nodes[j], &nodes[j+1], nil)
            newLevel = append(newLevel, *node)
        }

        nodes = newLevel
    }

    mTree := MerkleTree{&nodes[0]}

    return &mTree
}

// NewMerkleNode creates a new Merkle tree node
func NewMerkleNode(left, right *MerkleNode, data []byte) *MerkleNode {
    mNode := MerkleNode{}

    if left == nil && right == nil {
        hash := sha256.Sum256(data)
        mNode.Data = hash[:]
    } else {
        prevHashes := append(left.Data, right.Data...)
        hash := sha256.Sum256(prevHashes)
        mNode.Data = hash[:]
    }
```

```
    mNode.Left = left
    mNode.Right = right

    return &mNode
}
```

交易相关文件

transaction.go

```
package main

import (
    "bytes"
    "crypto/ecdsa"
    "crypto/elliptic"
    "crypto/rand"
    "crypto/sha256"
    "math/big"

    "encoding/gob"
    "encoding/hex"
    "fmt"
    "log"
    "strings"
)

const subsidy = 10

// Transaction represents a Bitcoin transaction
type Transaction struct {
    ID   []byte
```

```
    Vin  []TXInput
    Vout []TXOutput
}

// IsCoinbase checks whether the transaction is coinbase
func (tx Transaction) IsCoinbase() bool {
    return len(tx.Vin) == 1 && len(tx.Vin[0].Txid) == 0 && tx.Vin[0].Vout == -1
}

// Serialize returns a serialized Transaction
func (tx Transaction) Serialize() []byte {
    var encoded bytes.Buffer

    enc := gob.NewEncoder(&encoded)
    err := enc.Encode(tx)
    if err != nil {
        log.Panic(err)
    }

    return encoded.Bytes()
}

// Hash returns the hash of the Transaction
func (tx *Transaction) Hash() []byte {
    var hash [32]byte

    txCopy := *tx
    txCopy.ID = []byte{}

    hash = sha256.Sum256(txCopy.Serialize())

    return hash[:]
```

```
}

// Sign signs each input of a Transaction
func (tx *Transaction) Sign(privKey ecdsa.PrivateKey, prevTXs
map[string]Transaction) {
   if tx.IsCoinbase() {
      return
   }

   for _, vin := range tx.Vin {
      if prevTXs[hex.EncodeToString(vin.Txid)].ID == nil {
         log.Panic("ERROR: Previous transaction is not correct")
      }
   }

   txCopy := tx.TrimmedCopy()

   for inID, vin := range txCopy.Vin {
      prevTx := prevTXs[hex.EncodeToString(vin.Txid)]
      txCopy.Vin[inID].Signature = nil
      txCopy.Vin[inID].PubKey = prevTx.Vout[vin.Vout].PubKeyHash
      txCopy.ID = txCopy.Hash()
      txCopy.Vin[inID].PubKey = nil

      r, s, err := ecdsa.Sign(rand.Reader, &privKey, txCopy.ID)
      if err != nil {
         log.Panic(err)
      }
      signature := append(r.Bytes(), s.Bytes()...)

      tx.Vin[inID].Signature = signature
   }
```

```
}

// String returns a human-readable representation of a transaction
func (tx Transaction) String() string {
    var lines []string

    lines = append(lines, fmt.Sprintf("--- Transaction %x:", tx.ID))

    for i, input := range tx.Vin {

        lines = append(lines, fmt.Sprintf("     Input %d:", i))
        lines = append(lines, fmt.Sprintf("       TXID:      %x", input.Txid))
        lines = append(lines, fmt.Sprintf("       Out:       %d", input.Vout))
        lines = append(lines, fmt.Sprintf("       Signature: %x", input.Signature))
        lines = append(lines, fmt.Sprintf("       PubKey:    %x", input.PubKey))
    }

    for i, output := range tx.Vout {
        lines = append(lines, fmt.Sprintf("     Output %d:", i))
        lines = append(lines, fmt.Sprintf("       Value:  %d", output.Value))
        lines = append(lines, fmt.Sprintf("       Script: %x", output.PubKeyHash))
    }

    return strings.Join(lines, "\n")
}

// TrimmedCopy creates a trimmed copy of Transaction to be used in signing
func (tx *Transaction) TrimmedCopy() Transaction {
    var inputs []TXInput
    var outputs []TXOutput

    for _, vin := range tx.Vin {
```

```
        inputs = append(inputs, TXInput{vin.Txid, vin.Vout, nil, nil})
    }

    for _, vout := range tx.Vout {
        outputs = append(outputs, TXOutput{vout.Value, vout.PubKeyHash})
    }

    txCopy := Transaction{tx.ID, inputs, outputs}

    return txCopy
}

// Verify verifies signatures of Transaction inputs
func (tx *Transaction) Verify(prevTXs map[string]Transaction) bool {
    if tx.IsCoinbase() {
        return true
    }

    for _, vin := range tx.Vin {
        if prevTXs[hex.EncodeToString(vin.Txid)].ID == nil {
            log.Panic("ERROR: Previous transaction is not correct")
        }
    }

    txCopy := tx.TrimmedCopy()
    curve := elliptic.P256()

    for inID, vin := range tx.Vin {
        prevTx := prevTXs[hex.EncodeToString(vin.Txid)]
        txCopy.Vin[inID].Signature = nil
        txCopy.Vin[inID].PubKey = prevTx.Vout[vin.Vout].PubKeyHash
        txCopy.ID = txCopy.Hash()
```

```
            txCopy.Vin[inID].PubKey = nil

            r := big.Int{}
            s := big.Int{}
            sigLen := len(vin.Signature)
            r.SetBytes(vin.Signature[:(sigLen / 2)])
            s.SetBytes(vin.Signature[(sigLen / 2):])

            x := big.Int{}
            y := big.Int{}
            keyLen := len(vin.PubKey)
            x.SetBytes(vin.PubKey[:(keyLen / 2)])
            y.SetBytes(vin.PubKey[(keyLen / 2):])

            rawPubKey := ecdsa.PublicKey{curve, &x, &y}
            if ecdsa.Verify(&rawPubKey, txCopy.ID, &r, &s) == false {
               return false
            }
         }

         return true
      }

      // NewCoinbaseTX creates a new coinbase transaction
      func NewCoinbaseTX(to, data string) *Transaction {
         if data == "" {
            randData := make([]byte, 20)
            _, err := rand.Read(randData)
            if err != nil {
               log.Panic(err)
            }
```

```
        data = fmt.Sprintf("%x", randData)
    }

    txin := TXInput{[]byte{}, -1, nil, []byte(data)}
    txout := NewTXOutput(subsidy, to)
    tx := Transaction{nil, []TXInput{txin}, []TXOutput{*txout}}
    tx.ID = tx.Hash()

    return &tx
}

// NewUTXOTransaction creates a new transaction
func NewUTXOTransaction(from, to string, amount int, UTXOSet *UTXOSet) *Transaction
{
    var inputs []TXInput
    var outputs []TXOutput

    wallets, err := NewWallets()
    if err != nil {
        log.Panic(err)
    }
    wallet := wallets.GetWallet(from)
    pubKeyHash := HashPubKey(wallet.PublicKey)
    acc, validOutputs := UTXOSet.FindSpendableOutputs(pubKeyHash, amount)

    if acc < amount {
        log.Panic("ERROR: Not enough funds")
    }

    // Build a list of inputs
    for txid, outs := range validOutputs {
        txID, err := hex.DecodeString(txid)
```

```
            if err != nil {
                log.Panic(err)
            }

            for _, out := range outs {
                input := TXInput{txID, out, nil, wallet.PublicKey}
                inputs = append(inputs, input)
            }
        }

        // Build a list of outputs
        outputs = append(outputs, *NewTXOutput(amount, to))
        if acc > amount {
            outputs = append(outputs, *NewTXOutput(acc-amount, from)) // a change
        }

        tx := Transaction{nil, inputs, outputs}
        tx.ID = tx.Hash()
        UTXOSet.Blockchain.SignTransaction(&tx, wallet.PrivateKey)

        return &tx
    }
```

transaction_input.go

```
package main

import "bytes"

// TXInput represents a transaction input
type TXInput struct {
    Txid      []byte
    Vout      int
```

```
    Signature []byte
    PubKey    []byte
}

// UsesKey checks whether the address initiated the transaction
func (in *TXInput) UsesKey(pubKeyHash []byte) bool {
    lockingHash := HashPubKey(in.PubKey)

    return bytes.Compare(lockingHash, pubKeyHash) == 0
}
```

transaction_output.go

```
package main

import (
    "bytes"
    "encoding/gob"
    "log"
)

// TXOutput represents a transaction output
type TXOutput struct {
    Value      int
    PubKeyHash []byte
}

// Lock signs the output
func (out *TXOutput) Lock(address []byte) {
    pubKeyHash := Base58Decode(address)
    pubKeyHash = pubKeyHash[1 : len(pubKeyHash)-4]
    out.PubKeyHash = pubKeyHash
}
```

```go
// IsLockedWithKey checks if the output can be used by the owner of the pubkey
func (out *TXOutput) IsLockedWithKey(pubKeyHash []byte) bool {
   return bytes.Compare(out.PubKeyHash, pubKeyHash) == 0
}

// NewTXOutput create a new TXOutput
func NewTXOutput(value int, address string) *TXOutput {
   txo := &TXOutput{value, nil}
   txo.Lock([]byte(address))

   return txo
}

// TXOutputs collects TXOutput
type TXOutputs struct {
   Outputs []TXOutput
}

// Serialize serializes TXOutputs
func (outs TXOutputs) Serialize() []byte {
   var buff bytes.Buffer

   enc := gob.NewEncoder(&buff)
   err := enc.Encode(outs)
   if err != nil {
      log.Panic(err)
   }

   return buff.Bytes()
}
```

```
// DeserializeOutputs deserializes TXOutputs
func DeserializeOutputs(data []byte) TXOutputs {
   var outputs TXOutputs

   dec := gob.NewDecoder(bytes.NewReader(data))
   err := dec.Decode(&outputs)
   if err != nil {
      log.Panic(err)
   }

   return outputs
}
```

wallet.go

```
package main

import (
   "bytes"
   "crypto/ecdsa"
   "crypto/elliptic"
   "crypto/rand"
   "crypto/sha256"
   "log"

   "golang.org/x/crypto/ripemd160"
)

const version = byte(0x00)
const walletFile = "wallet.dat"
const addressChecksumLen = 4

// Wallet stores private and public keys
```

```go
type Wallet struct {
    PrivateKey ecdsa.PrivateKey
    PublicKey  []byte
}

// NewWallet creates and returns a Wallet
func NewWallet() *Wallet {
    private, public := newKeyPair()
    wallet := Wallet{private, public}

    return &wallet
}

// GetAddress returns wallet address
func (w Wallet) GetAddress() []byte {
    pubKeyHash := HashPubKey(w.PublicKey)

    versionedPayload := append([]byte{version}, pubKeyHash...)
    checksum := checksum(versionedPayload)

    fullPayload := append(versionedPayload, checksum...)
    address := Base58Encode(fullPayload)

    return address
}

// HashPubKey hashes public key
func HashPubKey(pubKey []byte) []byte {
    publicSHA256 := sha256.Sum256(pubKey)

    RIPEMD160Hasher := ripemd160.New()
    _, err := RIPEMD160Hasher.Write(publicSHA256[:])
```

```
    if err != nil {
        log.Panic(err)
    }
    publicRIPEMD160 := RIPEMD160Hasher.Sum(nil)

    return publicRIPEMD160
}

// ValidateAddress check if address if valid
func ValidateAddress(address string) bool {
    pubKeyHash := Base58Decode([]byte(address))
    actualChecksum := pubKeyHash[len(pubKeyHash)-addressChecksumLen:]
    version := pubKeyHash[0]
    pubKeyHash = pubKeyHash[1 : len(pubKeyHash)-addressChecksumLen]
    targetChecksum := checksum(append([]byte{version}, pubKeyHash...))

    return bytes.Compare(actualChecksum, targetChecksum) == 0
}

// Checksum generates a checksum for a public key
func checksum(payload []byte) []byte {
    firstSHA := sha256.Sum256(payload)
    secondSHA := sha256.Sum256(firstSHA[:])

    return secondSHA[:addressChecksumLen]
}

func newKeyPair() (ecdsa.PrivateKey, []byte) {
    curve := elliptic.P256()
    private, err := ecdsa.GenerateKey(curve, rand.Reader)
    if err != nil {
        log.Panic(err)
```

```
    }
    pubKey := append(private.PublicKey.X.Bytes(), private.PublicKey.Y.Bytes()...)

    return *private, pubKey
}
```

wallets.go

```
package main

import (
    "bytes"
    "crypto/elliptic"
    "encoding/gob"
    "fmt"
    "io/ioutil"
    "log"
    "os"
)

// Wallets stores a collection of wallets
type Wallets struct {
    Wallets map[string]*Wallet
}

// NewWallets creates Wallets and fills it from a file if it exists
func NewWallets() (*Wallets, error) {
    wallets := Wallets{}
    wallets.Wallets = make(map[string]*Wallet)

    err := wallets.LoadFromFile()

    return &wallets, err
```

```
}

// CreateWallet adds a Wallet to Wallets
func (ws *Wallets) CreateWallet() string {
   wallet := NewWallet()
   address := fmt.Sprintf("%s", wallet.GetAddress())

   ws.Wallets[address] = wallet

   return address
}

// GetAddresses returns an array of addresses stored in the wallet file
func (ws *Wallets) GetAddresses() []string {
   var addresses []string

   for address := range ws.Wallets {
      addresses = append(addresses, address)
   }

   return addresses
}

// GetWallet returns a Wallet by its address
func (ws Wallets) GetWallet(address string) Wallet {
   return *ws.Wallets[address]
}

// LoadFromFile loads wallets from the file
func (ws *Wallets) LoadFromFile() error {
   if _, err := os.Stat(walletFile); os.IsNotExist(err) {
      return err
```

```
    }

    fileContent, err := ioutil.ReadFile(walletFile)
    if err != nil {
        log.Panic(err)
    }

    var wallets Wallets
    gob.Register(elliptic.P256())
    decoder := gob.NewDecoder(bytes.NewReader(fileContent))
    err = decoder.Decode(&wallets)
    if err != nil {
        log.Panic(err)
    }

    ws.Wallets = wallets.Wallets

    return nil
}

// SaveToFile saves wallets to a file
func (ws Wallets) SaveToFile() {
    var content bytes.Buffer

    gob.Register(elliptic.P256())

    encoder := gob.NewEncoder(&content)
    err := encoder.Encode(ws)
    if err != nil {
        log.Panic(err)
    }
```

```
    err = ioutil.WriteFile(walletFile, content.Bytes(), 0644)
    if err != nil {
        log.Panic(err)
    }
}
```

命令行解析相关文件

cli.go

```
package main

import (
    "flag"
    "fmt"
    "log"
    "os"
)

// CLI responsible for processing command line arguments
type CLI struct{}

func (cli *CLI) printUsage() {
    fmt.Println("Usage:")
    fmt.Println("  createblockchain -address ADDRESS - Create a blockchain and send
genesis block reward to ADDRESS")
    fmt.Println("  createwallet - Generates a new key-pair and saves it into the
wallet file")
    fmt.Println("  getbalance -address ADDRESS - Get balance of ADDRESS")
    fmt.Println("  listaddresses - Lists all addresses from the wallet file")
    fmt.Println("  printchain - Print all the blocks of the blockchain")
```

```
    fmt.Println("  reindexutxo - Rebuilds the UTXO set")
    fmt.Println("  send -from FROM -to TO -amount AMOUNT - Send AMOUNT of coins from
FROM address to TO")
}

func (cli *CLI) validateArgs() {
    if len(os.Args) < 2 {
        cli.printUsage()
        os.Exit(1)
    }
}

// Run parses command line arguments and processes commands
func (cli *CLI) Run() {
    cli.validateArgs()

    getBalanceCmd := flag.NewFlagSet("getbalance", flag.ExitOnError)
    createBlockchainCmd := flag.NewFlagSet("createblockchain", flag.ExitOnError)
    createWalletCmd := flag.NewFlagSet("createwallet", flag.ExitOnError)
    listAddressesCmd := flag.NewFlagSet("listaddresses", flag.ExitOnError)
    printChainCmd := flag.NewFlagSet("printchain", flag.ExitOnError)
    reindexUTXOCmd := flag.NewFlagSet("reindexutxo", flag.ExitOnError)
    sendCmd := flag.NewFlagSet("send", flag.ExitOnError)

    getBalanceAddress := getBalanceCmd.String("address", "", "The address to get
balance for")
    createBlockchainAddress := createBlockchainCmd.String("address", "", "The
address to send genesis block reward to")
    sendFrom := sendCmd.String("from", "", "Source wallet address")
    sendTo := sendCmd.String("to", "", "Destination wallet address")
    sendAmount := sendCmd.Int("amount", 0, "Amount to send")
```

```
switch os.Args[1] {
case "getbalance":
   err := getBalanceCmd.Parse(os.Args[2:])
   if err != nil {
      log.Panic(err)
   }
case "createblockchain":
   err := createBlockchainCmd.Parse(os.Args[2:])
   if err != nil {
      log.Panic(err)
   }
case "createwallet":
   err := createWalletCmd.Parse(os.Args[2:])
   if err != nil {
      log.Panic(err)
   }
case "listaddresses":
   err := listAddressesCmd.Parse(os.Args[2:])
   if err != nil {
      log.Panic(err)
   }
case "printchain":
   err := printChainCmd.Parse(os.Args[2:])
   if err != nil {
      log.Panic(err)
   }
case "send":
   err := sendCmd.Parse(os.Args[2:])
   if err != nil {
      log.Panic(err)
   }
case "reindexutxo":
```

```
        err := reindexUTXOCmd.Parse(os.Args[2:])
        if err != nil {
            log.Panic(err)
        }
    default:
        cli.printUsage()
        os.Exit(1)
    }

    if getBalanceCmd.Parsed() {
        if *getBalanceAddress == "" {
            getBalanceCmd.Usage()
            os.Exit(1)
        }
        cli.getBalance(*getBalanceAddress)
    }

    if createBlockchainCmd.Parsed() {
        if *createBlockchainAddress == "" {
            createBlockchainCmd.Usage()
            os.Exit(1)
        }
        cli.createBlockchain(*createBlockchainAddress)
    }

    if createWalletCmd.Parsed() {
        cli.createWallet()
    }

    if listAddressesCmd.Parsed() {
        cli.listAddresses()
    }
```

```
    if printChainCmd.Parsed() {
        cli.printChain()
    }

    if reindexUTXOCmd.Parsed() {
        cli.reindexUTXO()
    }

    if sendCmd.Parsed() {
        if *sendFrom == "" || *sendTo == "" || *sendAmount <= 0 {
            sendCmd.Usage()
            os.Exit(1)
        }

        cli.send(*sendFrom, *sendTo, *sendAmount)
    }
}
```

cli_createblockchain.go

```
package main

import (
    "fmt"
    "log"
)

func (cli *CLI) createBlockchain(address string) {
    if !ValidateAddress(address) {
        log.Panic("ERROR: Address is not valid")
```

```
    }
    bc := CreateBlockchain(address)
    defer bc.db.Close()

    UTXOSet := UTXOSet{bc}
    UTXOSet.Reindex()

    fmt.Println("Done!")
}
```

cli_createwallet.go

```
package main

import "fmt"

func (cli *CLI) createWallet() {
    wallets, _ := NewWallets()
    address := wallets.CreateWallet()
    wallets.SaveToFile()

    fmt.Printf("Your new address: %s\n", address)
}
```

cli.gebalance.go

```
package main

import (
    "fmt"
    "log"
```

```
)

func (cli *CLI) getBalance(address string) {
   if !ValidateAddress(address) {
      log.Panic("ERROR: Address is not valid")
   }
   bc := NewBlockchain()
   UTXOSet := UTXOSet{bc}
   defer bc.db.Close()

   balance := 0
   pubKeyHash := Base58Decode([]byte(address))
   pubKeyHash = pubKeyHash[1 : len(pubKeyHash)-4]
   UTXOs := UTXOSet.FindUTXO(pubKeyHash)

   for _, out := range UTXOs {
      balance += out.Value
   }

   fmt.Printf("Balance of '%s': %d\n", address, balance)
}
```

cli_listaddress.go

```
package main

import (
   "fmt"
   "log"
)
```

```
func (cli *CLI) listAddresses() {
   wallets, err := NewWallets()
   if err != nil {
      log.Panic(err)
   }
   addresses := wallets.GetAddresses()

   for _, address := range addresses {
      fmt.Println(address)
   }
}
```

cli_printchain.go

```
package main

import (
   "fmt"
   "strconv"
)

func (cli *CLI) printChain() {
   bc := NewBlockchain()
   defer bc.db.Close()

   bci := bc.Iterator()

   for {
      block := bci.Next()

      fmt.Printf("============ Block %x ============\n", block.Hash)
```

```
            fmt.Printf("Prev. block: %x\n", block.PrevBlockHash)
            pow := NewProofOfWork(block)
            fmt.Printf("PoW: %s\n\n", strconv.FormatBool(pow.Validate()))
            for _, tx := range block.Transactions {
                fmt.Println(tx)
            }
            fmt.Printf("\n\n")

            if len(block.PrevBlockHash) == 0 {
                break
            }
        }
    }
```

cli_reindexutxo.go

```
package main

import "fmt"

func (cli *CLI) reindexUTXO() {
    bc := NewBlockchain()
    UTXOSet := UTXOSet{bc}
    UTXOSet.Reindex()

    count := UTXOSet.CountTransactions()
    fmt.Printf("Done! There are %d transactions in the UTXO set.\n", count)
}
```

cli_send.go

```
package main

import (
   "fmt"
   "log"
)

func (cli *CLI) send(from, to string, amount int) {
   if !ValidateAddress(from) {
      log.Panic("ERROR: Sender address is not valid")
   }
   if !ValidateAddress(to) {
      log.Panic("ERROR: Recipient address is not valid")
   }

   bc := NewBlockchain()
   UTXOSet := UTXOSet{bc}
   defer bc.db.Close()

   tx := NewUTXOTransaction(from, to, amount, &UTXOSet)
   cbTx := NewCoinbaseTX(from, "")
   txs := []*Transaction{cbTx, tx}

   newBlock := bc.MineBlock(txs)
   UTXOSet.Update(newBlock)
   fmt.Println("Success!")
}
```

其他程序

utils.go

```
package main

import (
   "bytes"
   "crypto/elliptic"
   "encoding/gob"
   "fmt"
   "io/ioutil"
   "log"
   "os"
)

// Wallets stores a collection of wallets
type Wallets struct {
   Wallets map[string]*Wallet
}

// NewWallets creates Wallets and fills it from a file if it exists
func NewWallets() (*Wallets, error) {
   wallets := Wallets{}
   wallets.Wallets = make(map[string]*Wallet)

   err := wallets.LoadFromFile()

   return &wallets, err
}

// CreateWallet adds a Wallet to Wallets
func (ws *Wallets) CreateWallet() string {
```

```
    wallet := NewWallet()
    address := fmt.Sprintf("%s", wallet.GetAddress())

    ws.Wallets[address] = wallet

    return address
}

// GetAddresses returns an array of addresses stored in the wallet file
func (ws *Wallets) GetAddresses() []string {
    var addresses []string

    for address := range ws.Wallets {
        addresses = append(addresses, address)
    }

    return addresses
}

// GetWallet returns a Wallet by its address
func (ws Wallets) GetWallet(address string) Wallet {
    return *ws.Wallets[address]
}

// LoadFromFile loads wallets from the file
func (ws *Wallets) LoadFromFile() error {
    if _, err := os.Stat(walletFile); os.IsNotExist(err) {
        return err
    }

    fileContent, err := ioutil.ReadFile(walletFile)
    if err != nil {
```

```
        log.Panic(err)
    }

    var wallets Wallets
    gob.Register(elliptic.P256())
    decoder := gob.NewDecoder(bytes.NewReader(fileContent))
    err = decoder.Decode(&wallets)
    if err != nil {
        log.Panic(err)
    }

    ws.Wallets = wallets.Wallets

    return nil
}

// SaveToFile saves wallets to a file
func (ws Wallets) SaveToFile() {
    var content bytes.Buffer

    gob.Register(elliptic.P256())

    encoder := gob.NewEncoder(&content)
    err := encoder.Encode(ws)
    if err != nil {
        log.Panic(err)
    }

    err = ioutil.WriteFile(walletFile, content.Bytes(), 0644)
    if err != nil {
        log.Panic(err)
```

```
    }
}
```

utxo_set.go

```
package main

import (
    "encoding/hex"
    "log"

    "github.com/boltdb/bolt"
)

const utxoBucket = "chainstate"

// UTXOSet represents UTXO set
type UTXOSet struct {
    Blockchain *Blockchain
}

// FindSpendableOutputs finds and returns unspent outputs to reference in inputs
func (u UTXOSet) FindSpendableOutputs(pubkeyHash []byte, amount int) (int,
map[string][]int) {
    unspentOutputs := make(map[string][]int)
    accumulated := 0
    db := u.Blockchain.db

    err := db.View(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(utxoBucket))
        c := b.Cursor()

        for k, v := c.First(); k != nil; k, v = c.Next() {
```

```
            txID := hex.EncodeToString(k)
            outs := DeserializeOutputs(v)

            for outIdx, out := range outs.Outputs {
               if out.IsLockedWithKey(pubkeyHash) && accumulated < amount {
                  accumulated += out.Value
                  unspentOutputs[txID] = append(unspentOutputs[txID], outIdx)
               }
            }
         }

         return nil
      })
      if err != nil {
         log.Panic(err)
      }

      return accumulated, unspentOutputs
   }

   // FindUTXO finds UTXO for a public key hash
   func (u UTXOSet) FindUTXO(pubKeyHash []byte) []TXOutput {
      var UTXOs []TXOutput
      db := u.Blockchain.db

      err := db.View(func(tx *bolt.Tx) error {
         b := tx.Bucket([]byte(utxoBucket))
         c := b.Cursor()

         for k, v := c.First(); k != nil; k, v = c.Next() {
            outs := DeserializeOutputs(v)
```

```
            for _, out := range outs.Outputs {
                if out.IsLockedWithKey(pubKeyHash) {
                    UTXOs = append(UTXOs, out)
                }
            }
        }

        return nil
    })
    if err != nil {
        log.Panic(err)
    }

    return UTXOs
}

// CountTransactions returns the number of transactions in the UTXO set
func (u UTXOSet) CountTransactions() int {
    db := u.Blockchain.db
    counter := 0

    err := db.View(func(tx *bolt.Tx) error {
        b := tx.Bucket([]byte(utxoBucket))
        c := b.Cursor()

        for k, _ := c.First(); k != nil; k, _ = c.Next() {
            counter++
        }

        return nil
    })
    if err != nil {
```

```
            log.Panic(err)
        }

        return counter
    }

    // Reindex rebuilds the UTXO set
    func (u UTXOSet) Reindex() {
        db := u.Blockchain.db
        bucketName := []byte(utxoBucket)

        err := db.Update(func(tx *bolt.Tx) error {
            err := tx.DeleteBucket(bucketName)
            if err != nil && err != bolt.ErrBucketNotFound {
                log.Panic(err)
            }

            _, err = tx.CreateBucket(bucketName)
            if err != nil {
                log.Panic(err)
            }

            return nil
        })
        if err != nil {
            log.Panic(err)
        }

        UTXO := u.Blockchain.FindUTXO()

        err = db.Update(func(tx *bolt.Tx) error {
            b := tx.Bucket(bucketName)
```

```
      for txID, outs := range UTXO {
         key, err := hex.DecodeString(txID)
         if err != nil {
            log.Panic(err)
         }

         err = b.Put(key, outs.Serialize())
         if err != nil {
            log.Panic(err)
         }
      }

      return nil
   })
}

// Update updates the UTXO set with transactions from the Block
// The Block is considered to be the tip of a blockchain
func (u UTXOSet) Update(block *Block) {
   db := u.Blockchain.db

   err := db.Update(func(tx *bolt.Tx) error {
      b := tx.Bucket([]byte(utxoBucket))

      for _, tx := range block.Transactions {
         if tx.IsCoinbase() == false {
            for _, vin := range tx.Vin {
               updatedOuts := TXOutputs{}
               outsBytes := b.Get(vin.Txid)
               outs := DeserializeOutputs(outsBytes)
```

```
            for outIdx, out := range outs.Outputs {
                if outIdx != vin.Vout {
                    updatedOuts.Outputs = append(updatedOuts.Outputs, out)
                }
            }

            if len(updatedOuts.Outputs) == 0 {
                err := b.Delete(vin.Txid)
                if err != nil {
                    log.Panic(err)
                }
            } else {
                err := b.Put(vin.Txid, updatedOuts.Serialize())
                if err != nil {
                    log.Panic(err)
                }
            }

        }
    }

    newOutputs := TXOutputs{}
    for _, out := range tx.Vout {
        newOutputs.Outputs = append(newOutputs.Outputs, out)
    }

    err := b.Put(tx.ID, newOutputs.Serialize())
    if err != nil {
        log.Panic(err)
    }
}
```

```
        return nil
    })
    if err != nil {
        log.Panic(err)
    }
}
```

命令行所有操作


1920 × 1026
Microsoft Windows [版本 10.0.18363.1316]
(c) 2019 Microsoft Corporation. 保留所有权利.
C:\Users\Jerry>cd /d E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createwallet
Your new address: 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
No existing blockchain found. Create one first.
exit status 1
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ createblockchain -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6
Done!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 10
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ listaddresses
1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
1FBWEkmfUr6kvuMM2gZJu9fHsolPgFhQn3
1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
1FXDQMRLXhFZeuntZKY9yDQ9uyUNbFgp6i
1MMsuAlTHnCmgSnrNiBKXyCaBwjuTUzsu
1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
164pplPUQ3ykrictxQrte9RJ5eHEf27Pdt
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL -to 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa -amount 8
000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4
Success!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL -to 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7 -amount 1
000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b
Success!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 1
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoxv6TcUjpCJTnGQxSgf72NHZa n
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 8
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7': 1
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa -to 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7 -amount 7
000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186
Success!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 1
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 1
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7': 8
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 1
000000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d
Success!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 1
000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033
Success!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7 -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 8
000000065fab8754aa3938724297c70cadb95e92fdccc95fc42371cc8622825d5
Success!
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNePlm46zSu7': 0
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 10
E:\Jerry\My University\大学\课程\第三学期\go语言与区块链\期末设计\My_Blockchain>go run ./ printchain
========== Block 000000065fab8754aa3938724297c70cadb95e92fdccc95fc42371cc8622825d5 ==========
Prev. block: 000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033
PoW: true
--- Transaction 175f19bec5fc610796bb8d88d61e128af591aec4ff2c5e84ebf9632cbda01c7a:
Input 0:
TXID: 9587e41c06ddfac3a5623138f42bc34dcalbf5efe8eb71c2ebd977f51e78d794
Out: 0

Microsoft Windows [版本 10.0.18363.1316]

(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\Jerry>cd /d E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计\My_Blockchain

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计

```
\My_Blockchain>go run ./ createwallet
Your new address: 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ createwallet
Your new address: 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ createwallet
Your new address: 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ createwallet
Your new address: 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
No existing blockchain found. Create one first.
exit status 1

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ createblockchain -address
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6

Done!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 10
```

```
E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ listaddresses
1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
1FBWEkmfUr6kvuMM2gZJu9fHso1PgFhQn3
1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
1FXDQMRLXhFZeuntZKY9yDQ9uyUNbFgp6i
1MMsuA1THnCmgSnrNiBKXyCaBwjuf7Uzsu
1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
164pp1PUQ3ykrictxQrte9RJ5eHEf27Fdt
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
-to 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa -amount 8
000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4
```

```
Success!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
-to 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7 -amount 1
000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b

Success!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 1

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1AGuGAjBoxv6TcUjpCJTnGQxSgf72NHZa                    n
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 8

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 1

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ send -from 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
-to 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7 -amount 7
```

000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186

Success!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 1

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 1

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 8

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计\My_Blockchain>go run ./ getbalance -address 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计\My_Blockchain>go run ./ send -from 1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL -to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 1
00000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d

Success!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计

```
\My_Blockchain>go run ./ send -from 1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
-to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 1
000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033

Success!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ send -from 1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
-to 1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve -amount 8
00000065fab8754aa3938724297c70cadb95e92fdccc95fc42371cc8622825d5

Success!

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL
Balance of '1JGQkT1EabQnw4Ayc7PusPmWvaVkfrAdSL': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa
Balance of '1AGuGAjBoSxv6TcUjpCJTnGQxSgf72NHZa': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7
Balance of '1F2uQPKvztt2bPji9Jb8tfBNeP1m46zSu7': 0

E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ getbalance -address
1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve
Balance of '1PuGH8E87z9DQp7nBnE89tFNiH4euvz7ve': 10
```

```
E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>go run ./ printchain
============                                                    Block
00000065fab8754aa3938724297c70cadb95e92fdccc95fc42371cc8622825d5
============
Prev.                                                           block:
000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033
PoW: true

---                                                             Transaction
175f19bec5fc610796bb8d88d61e128af591aec4ff2c5e84ebf9632cbda01c7a:
     Input 0:
       TXID:
9587e41c06ddfac3a5623138f42bc34dca1bf5efe8eb71c2ebd977f51e78d794
       Out:       0
       Signature:
bf5b80abe9ffd07ab9d7e52f8b95e28830443ae7e6295c459913e63a5bcf5e3248e91
73dc5cce5fdb9c3a09e58bf3549b9b0eb9116b43edc20c1fe365b1da361
       PubKey:
daf35edd085eb050e2961bd547f17acb5f016602a0a3c86f3c78fd0c1b20dc1310c82
69baf10b322383cba744424ea7089e13f048bac37ab38d2ac2798f82751
     Input 1:
       TXID:
107cdc7f1840f116c7d292970215e819ddc06333b1c55c26464f99ea758b357d
       Out:       0
       Signature:
607ad77f592deb2b5b802d7679f9283c717cab2d3df09b95ed6bbe24162ee2a1dac41
832f481aa04245b94d307eecf0ea440ffaa29d3695c71af48e3663faac9
       PubKey:
daf35edd085eb050e2961bd547f17acb5f016602a0a3c86f3c78fd0c1b20dc1310c82
69baf10b322383cba744424ea7089e13f048bac37ab38d2ac2798f82751
```

```
    Output 0:
      Value:  8
      Script: fb352694109e8ae2dd47e1189e4bcb0e5fcac8cf


============                                                    Block
000000207fe02b05d712d474fe0a1bd6377daca69eb5f60658d329f5305eb033
============
Prev.                                                           block:
00000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d
PoW: true

---                                                             Transaction
0780efe0263f6fb0a1ac0b9137425c5ba5e24337f2ae74aa55447fdd5cf885a5:
    Input 0:
      TXID:
9587e41c06ddfac3a5623138f42bc34dca1bf5efe8eb71c2ebd977f51e78d794
      Out:       1
      Signature:
2edc61700f0e00874111011fd80cd08e6999805e447e7cacb664bd34b14fdaf941ef1
c88e62b0c8b4d35eab4d3af600b8d6a28d3b5a1f61d50f2ad7e12de5da4
      PubKey:
4a26b8be612c2d8b84397f176168628b2392d16b8683c05b9198c7c194ace61f6af9f
b85da432c67dcea45175cb72c2ef44e8eda3f04d4fa6b08ab00d1e1ea94
    Output 0:
      Value:  1
      Script: fb352694109e8ae2dd47e1189e4bcb0e5fcac8cf


============                                                    Block
00000089b0c82de2830c1e6ab7d544903311325e3c6b1848090eafbfc87d625d
============
```

```
Prev.                                                          block:
000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186
PoW: true

---                                                          Transaction
dbb7a7fdd1ba3be6a2602199ba9abc12ba6953df9c479d2d91d90631e3532c4a:
     Input 0:
       TXID:
107cdc7f1840f116c7d292970215e819ddc06333b1c55c26464f99ea758b357d
       Out:       1
       Signature:
ba17f1648a999075f2b0d4e3b8aed83623e95849eefb6ad2c77dc64b879d3da3c53a3
dd6fb1e51827739c29fd6bdb056616d0dc465cb4c79af142c1a65f685c0
       PubKey:
a7e85e0623eb5ba93a77246705db6fcd15b571f21ca5839e7920ff5bb69a320088977
1143f2c4f971d55b7c559294647f6fbb656ad54ab4974a04499cb40af30
     Output 0:
       Value:  1
       Script: fb352694109e8ae2dd47e1189e4bcb0e5fcac8cf


============                                                   Block
000000339d0f394d3da1279904e11c4cd3df2c88a35dff17cbc73ee66eed4186
============
Prev.                                                          block:
000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b
PoW: true

---                                                          Transaction
9587e41c06ddfac3a5623138f42bc34dca1bf5efe8eb71c2ebd977f51e78d794:
     Input 0:
       TXID:
```

918b1c5bc438d6a042811bd08effb61afb983fd80c039a3ff37faa3af05ce45f

Out: 0

Signature:

6c34a4c293f280f64e4a6113fabb2ab9dbb0bd7befdfa6831e5205f22f505c1f76ca4

49f3389d829970c05f7170a054e2df60651541a4c5bab5dd7dabd444a16

PubKey:

4a26b8be612c2d8b84397f176168628b2392d16b8683c05b9198c7c194ace61f6af9f

b85da432c67dcea45175cb72c2ef44e8eda3f04d4fa6b08ab00d1e1ea94

Output 0:

Value: 7

Script: 99edebc0fdf519b5d863fa7f92e72c1e530f1736

Output 1:

Value: 1

Script: 65baff48208d3accdbcabe47f1ae76b996275d07

============ Block

000000a20b27253e821b10ec56b5dfeeba6a6d8240f860ac376b036c3ea9d90b

============

Prev. block:

000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4

PoW: true

--- Transaction

107cdc7f1840f116c7d292970215e819ddc06333b1c55c26464f99ea758b357d:

Input 0:

TXID:

918b1c5bc438d6a042811bd08effb61afb983fd80c039a3ff37faa3af05ce45f

Out: 1

Signature:

b65e32f50f94f438a9e8e9468099f1dc3fd71a1853ee2381e6a24421c8992d158a989

341d03364ded19c1a449cdc8c53c87a0f1871c8950736082b82fef95f3c

PubKey:
a7e85e0623eb5ba93a77246705db6fcd15b571f21ca5839e7920ff5bb69a320088977
1143f2c4f971d55b7c559294647f6fbb656ad54ab4974a04499cb40af30
Output 0:
Value: 1
Script: 99edebc0fdf519b5d863fa7f92e72c1e530f1736
Output 1:
Value: 1
Script: bd6433d7a4f3e01b2416dcc7828f63064511b697

============ Block
000000e77daa08e3b592d6cf9a30d4774e423e01acc8a1aafcedd9e5c11670b4
============
Prev. block:
000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6
PoW: true

--- Transaction
918b1c5bc438d6a042811bd08effb61afb983fd80c039a3ff37faa3af05ce45f:
Input 0:
TXID:
565dc3b93c0e62689475e13b957453009e2289b02d94a32d0d75ea144d8dd07f
Out: 0
Signature:
3d191788c43d0ba2a2025691c1f6f57caba4448ad7b8c921429728ee86e5f0ba8356e
781857116865290dfcd03d982a8a68c0614899e7003d1c0347aa483e2df
PubKey:
a7e85e0623eb5ba93a77246705db6fcd15b571f21ca5839e7920ff5bb69a320088977
1143f2c4f971d55b7c559294647f6fbb656ad54ab4974a04499cb40af30
Output 0:
Value: 8

```
      Script: 65baff48208d3accdbcabe47f1ae76b996275d07
    Output 1:
      Value:  2
      Script: bd6433d7a4f3e01b2416dcc7828f63064511b697


============                                                    Block
000000a730a7302b9e39d910914b4740b02006314cdf0316fdee408ce5fa6da6
============
Prev. block:
PoW: true

---                                                             Transaction
565dc3b93c0e62689475e13b957453009e2289b02d94a32d0d75ea144d8dd07f:
    Input 0:
      TXID:
      Out:       -1
      Signature:
      PubKey:
5468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6
e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73
    Output 0:
      Value:  10
      Script: bd6433d7a4f3e01b2416dcc7828f63064511b697


E:\Jerry\My University\大学\课程\第三学期\go 语言与区块链\期末设计
\My_Blockchain>
```