**POS算法实验指导书**

**一、实验名称**

基于 PoS 共识机制的区块链实验

**二、实验内容**

本实验旨在让学生通过构建一个基于 PoS 共识机制的模拟区块链系统，了解区块链的基本原理和 PoS 共识机制的实现过程。具体内容包括：

理解 PoS 共识机制的原理和实现方式

实现一个简单的 PoS 共识算法

**三、实验环境**

本实验需要使用以下工具和环境：

Go 语言开发环境（示例代码的go版本为1.20，不建议go版本低于1.13）

具备基本的命令行和编程能力的计算机操作系统

**四、算法描述**

PoS（Proof of Stake）共识机制是一种基于代币持有量来确定区块链网络共识的算法。相比于 PoW（Proof of Work）共识机制，PoS 具有更高的效率和更低的能源消耗。使用Pos机制的区块链共识主要流程如下：

1、初始状态下，网络中的每个节点都需要拥有一定数量的数字资产作为抵押。这些数字资产将被用作随机选择记账节点的依据。

2、当需要生成一个新的区块时，网络会根据参与者的抵押数量来选择出共识节点（也称为验证节点或记账人），并由共识节点来完成区块的验证和打包。

3、其他节点将对新区块进行验证，确保其中的交易合法且不与其他已经存在的交易冲突。

4、共识节点会获得一定的奖励作为激励，而抵押的代币则会被锁定一段时间（称为冻结期），以确保共识节点不会恶意攻击网络。

PoS共识算法实现：

PoS 共识算法中引入了“币龄”的概念，相关定义如下：

币龄（coinAge）是指持币数量（coins）与持币时间 （holdTime）的乘积： coinAge=

coins × holdTime

PoS共识算法给定一个全网统一的难度值 D，以及新打包进区块的元数据

tradeData，寻找满足条件的记数器 timeCounter， 使得：

SHA256(SHA256(tradeData|timeCounter))<D×coinAge，找到则挖矿成功。

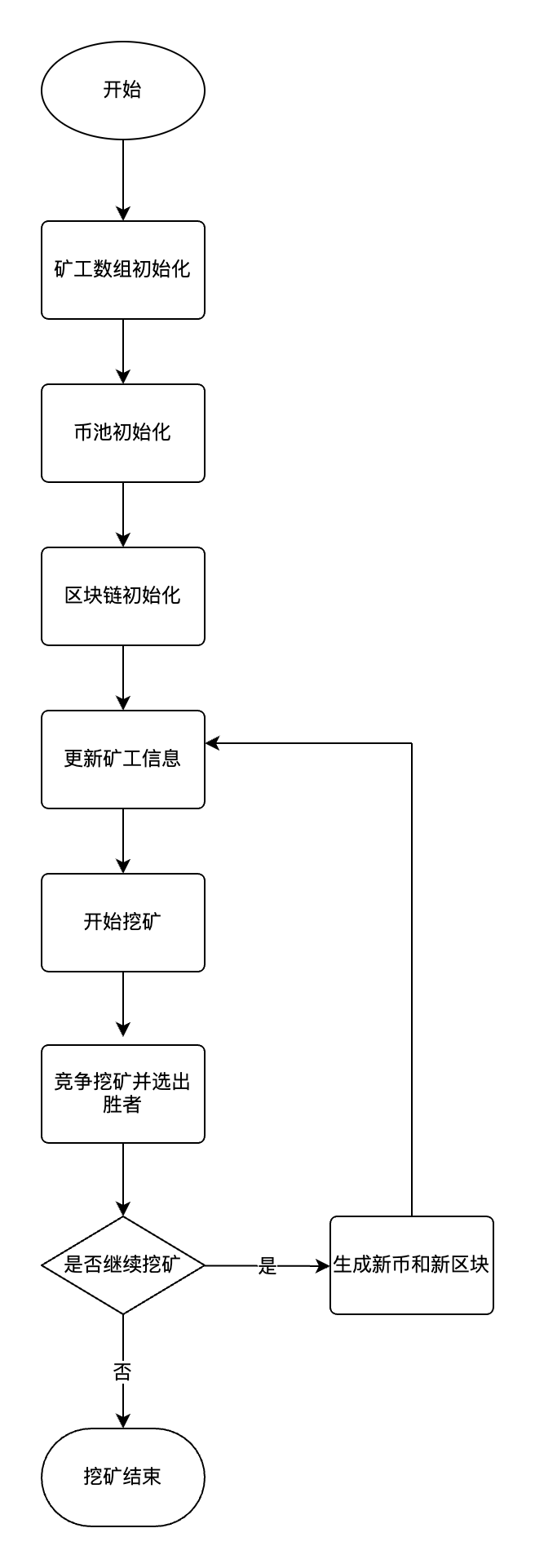
PoS 共识算法的记账规则与 Pow 共识算法基本相似，但 PoS共识算法不需要矿

工枚举所有的随机数 Nonce，而是在 1s内只允许一次哈希值，大大减轻了计算工作

量，从而减缓了算力竞争带来的资源消耗。

1. **实验过程**

**实验流程：**



**①矿工数组初始化**

// 初始化Miners数组的函数，调用AddMiner函数，生成一个Miner，然后将其添加到Miners数组中

func InitMiners() []Miner {

miner := createMiner()

Miners := []Miner{\*miner}

return Miners

}

**②币池初始化**

// 初始化coins数组函数，调用NewCoin函数，生成一个coin，然后将其添加到coins数组中

func InitCoins(Miners []Miner) []Coin {

coin := NewCoin(0, Miners)

Coins := []Coin{coin}

return Coins

}

**③区块链初始化**

// 初始化函数，生成创世区块，并添加到区块链中

func InitBlockChain(Miners []Miner, Coins []Coin) []Block {

var bc []Block

bc = append(bc, GenesisBlock(Miners, Coins))

return bc

}

**④更新矿工信息**

// 更新Miners数组函数，传入Coins数组和Miners数组，遍历Coins数组，将Coins数组中的币的矿工序号与Miners数组中的矿工序号相同的矿工的币龄加上（现在的时间-Coin的时间戳）\*Coin的数量

func UpdateMiners(Coins \*[]Coin, Miners \*[]Miner) []Miner {

for i := 0; i < len(\*Coins); i++ {

index := (\*Coins)[i].MinerIndex

(\*Miners)[index].coinAge += (time.Now().Unix() - (\*Coins)[i].Time) \* (\*Coins)[i].Num

(\*Coins)[i].Time = time.Now().Unix()

}

return \*Miners

}

**⑤pos挖矿**

// 函数名：Pos,传入Miners数组，当前难度值Dif和一个string类型变量tradeData，内设一个int变量timeCounter, 从0递增到Intmax，

//hash值为SHA256(SHA256(tradeData|timeCounter)),循环内遍历Miners数组，目标值target=Dif乘当前Miner的币龄，

//要求hash小于target，返回满足要求的第一个Miner的序号并清空这个Miner的币龄，一旦满足要求则退出整个循环

func Pos(Miners Miner, Dif int64, tradeData string) bool {

var timeCounter int

var realDif int64

realDif = int64(MinProbably)

if realDif +Dif\*Miners.coinAge > int64(MaxProbably) {

realDif = MaxProbably

} else {

realDif += Dif \* Miners.coinAge

}

target := big.NewInt(1)

// 数据长度为8位

//需求：需要满足前两位为0，才能解决问题

//1 \* 2 << (8-2) = 64

// 0100 0000

// 00xx xxxx

// 32 \* 8

target.Lsh(target, uint(realDif))

for timeCounter = 0; timeCounter < INT64\_MAX; timeCounter++ {

hash := sha256.Sum256([]byte(tradeData + string(timeCounter)))

hash = sha256.Sum256(hash[:])

var hashInt big.Int

hashInt.SetBytes(hash[:])

if hashInt.Cmp(target) == -1 {

return true

}

}

return false

}

**⑥竞争挖矿（时间最短者胜出）**

func CorrectMiner(Miners \*[]Miner, Dif int64, tradeData string) int{

var minTime int64 = INT64\_MAX

var correctMiner int

var MinerData []MinerTime

for i := 0; i < len(\*Miners); i++ {

start = time.Now().UnixNano()

//最小持币量为2才能挖矿

time.Sleep(1)

if (\*Miners)[i].num >= 2 {

success := Pos((\*Miners)[i], Dif, tradeData)

if success ==true {

end = time.Now().UnixNano()

MinerDataDemo := MinerTime{

minerIndex: i,

totalTime: end - start,

}

AddMinerData(&MinerData, &MinerDataDemo)

}

}

}

if MinerData != nil {

fmt.Println(MinerData)

for j, \_ := range MinerData{

if MinerData[j].totalTime < minTime {

minTime = MinerData[j].totalTime

correctMiner = MinerData[j].minerIndex

}

}

(\*Miners)[correctMiner].coinAge = 0

return correctMiner

}

return -1

**⑦生成新币**

// 传入新coin和coins数组，将其添加到coins数组中并保存，无返回值

func AddCoin(coin Coin, Coins \*[]Coin) {

\*Coins = append(\*Coins, coin)

}

**⑧打印本轮挖矿完后矿工信息**

//传入Miners数组，打印矿工数组每个矿工信息的函数

func PrintMiners(Miners []Miner) {

for i := 0; i <= len(Miners)-1; i++ {

fmt.Println("Miner", i, ":", hex.EncodeToString(Miners[i].addr), Miners[i].num, Miners[i].coinAge)

}

}

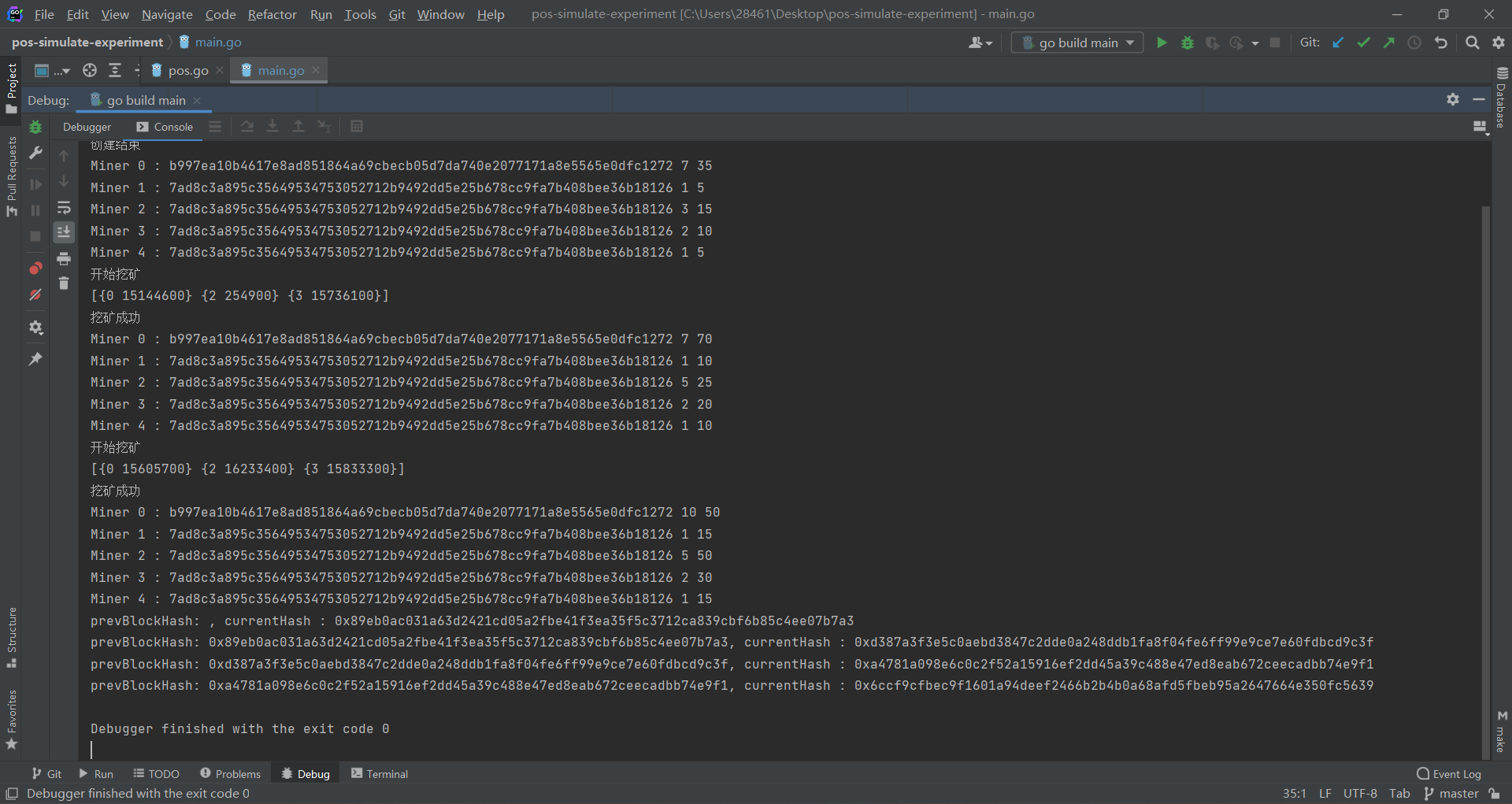
**⑨挖矿结束**

**Note:每轮挖矿完延迟5s（time.Sleep(5\*time.Second)），用以计算币龄。**

1. **实验结果**

**测试示例：初始化5个矿工并随机给他们发放初试币，进行两轮挖矿。打印出每轮挖矿各矿工所用时间并打印出挖矿完各矿工持币数和币龄。**

**测试结果样例：**

****

**七、实验心得**