实验八：使用 Go 语言编程实现常见共识机制

一、实验目标

通过自主设计实验，体验典型共识机制的原理及运作过程，感受区块链系统的奥妙。实验内容主要分为：

1. 实验 1：工作量证明（PoW）算法实现
2. 实验 2：权益证明（PoS）算法实现
3. 实验 3：委托权益证明（DPoS）算法实现

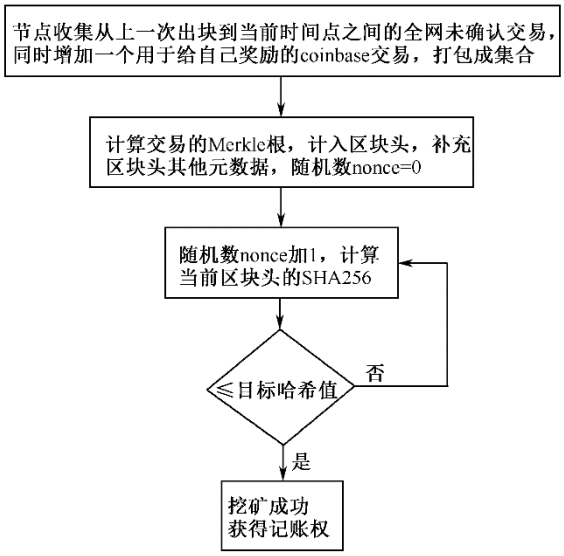
二、预备知识

工作量证明算法 PoW、权益证明算法 PoS、委托人证明算法 DPoS 和实用拜占庭容错 PBFT 算法是常见的共识算法，这些都是在学术界和工业界广泛应用的共识算法，经过了时间的检测，并且不断改进，在不同的应用场景下保证了区块链系统的安全可靠运行。当然，也有很多新兴的区块链共识算法，如在物联网领域兴起的基于有向无环图 DAG 的缠结共识算法。

1．工作量证明 PoW

PoW（Proof of Work）是中本聪在其比特币奠基性论文中提出来的用于比特币的一种共识机制，也是最早在区块链得到大范围应用的一种共识机制。PoW 已经安全运行了超过十年时间，事实证明，PoW 是安全可靠的。PoW 的原理是一方提出一个难以计算却能够被简单验证的数学难题，其他人能够快速验证提交人运算的准确性，从而判断该提交人完成了在这个过程中的大量工作。

PoW 共识机制的设计正是效仿了上述的设计，每个节点通过计算求解一个运算复杂但是验证相对容易的 SHA256 数学难题，这个计算过程就是我们最常听到的挖矿的过程。最早计算出该数学难题的挖矿节点将能够获得区块链的记账出块权利，并且挖矿节点能够获得一定的比特币和交易费作为其挖矿奖励。区块链系统可以控制挖矿难度来控制出块速度，如比特币网络的出块速度大致为 10 分钟一个。PoW 共识机制能够抵抗 51%攻击，也就是说，攻击者的算力要达到整个网络中总算力的 51%才能够成功攻击。图 9-1 是比特币 PoW 共识机制的运行原理。



2．权益证明 PoS

比特币采用的 PoW 共识机制有着很好的安全性，但是其采用的哈希解谜的挖矿过程有着非常庞大的电力消耗。为了节省资源等一系列需求，研究者在尝试用虚拟挖矿代替实际的计算挖矿。虚拟挖矿是一种只需要少量计算资源就能够代替全网挖矿的挖矿方式，其中本节 PoS（Proof of Stake）共识机制就是一种非常具有代表性的虚拟挖矿方式。

PoS 使用权益证明的方式替代工作量证明，这样会使系统中具有最高权益的节点而不是最高算力的节点具有记账权，权益是通过币龄来计算的。币龄是一个节点对特定数量的币的所有权，是通过交易金额（币）乘以交易的币在账上存留的时间（天）得出的权力。币龄越大，节点越有可能获得记账出块的权力。因为不用全网挖矿，PoS 共识机制不需消耗大量的能源。

3．委托权益证明 DPoS

授权股份证明是通过一个去中心化的民主选举解决了 PoW 和 PoS 存在的问题，每个币都是一票，持币人可以将自己的票投给信任的节点，系统会根据票数统计得票最多的几个节点，称为系统受托人。

采用 DPoS（Delegated Proof of Stake）共识机制的有比特股。比特股引入见证人的概念，每个比特股持股人都能够进行投票选举见证人，从总票数中选取票数最多的 N 个候选人成为见证人，见证人能够打包区块进行出块，并且见证人的个数 N 必须满足一半的投票人相信 N 已经充分去中心化。候选人的名单会周期性更新。见证人会按照一个随机顺序生成区块。

三、环境搭建

开放实验不限制实验环境，但推荐读者使用 Go 语言完成本节实验。

四、实验内容

实验 1　PoW 共识的实现

实验 1.1 区块结构

在区块链中，真正存储有效信息的是区块（block）。而在比特币中，真正有价值的信息就是交易（transaction）。实际上，交易信息是所有加密货币的价值所在。除此以外，区块还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。

由于我们要实现的是一条简化版的区块链，因此区块中暂时将仅包含以下信息：

```
 type block struct {
//上一个区块的 Hash
Lasthash string
//本区块 Hash
Hash string
//区块存储的数据（比如比特币 UTXO 模型 则此处可用于存储交易）
Data string
//时间戳
Timestamp string
//区块高度
Height int
//难度值
DiffNum uint
//随机数
Nonce int64
}
```

我们的 Data 实际是区块链中的”Transaction”字段，目前暂时不会涉及太过复

杂的结构，只需要知道是一串字符信息即可。而最后的 Hash 字段，则用来记录当前块的哈希。哈希计算，是区块链一个非常重要的部分。正是由于它，才保证了区块链的安全。计算一个哈希，是在计算上非常困难的一个操作。即使在高速电脑上，也要耗费很多时间 (这就是为什么人们会购买 GPU，FPGA，ASIC 来挖比特币) 。这是一个架构上有意为之的设计，它故意使得加入新的区块十分困难，继而保证区块一旦被加入以后，就很难再进行修改。

实验 1.2 设计共识

挖矿的伪代码如下（需要同学们完善代码）：

```
    //区块挖矿（通过自身递增 nonce 值计算 hash）
func mine(data string) block {
    if len(blockchain) < 1 {
        log.Panic("还未生成创世区块！")
    }
    lastBlock := blockchain[len(blockchain)-1]
    //制造一个新的区块
    nowBlock:= new(block)
    // 产生一个 nonce
    Nonce:=generateNonce()
    //根据挖矿难度值计算一个符合 hash
    While(true){
        If(符合要求){
            break;
        }
    }
    return *newBlock
}
```

main.go 如下：

```
 func main() {
    //制造一个创世区块
    genesisBlock := new(block)
    genesisBlock.Timestamp = time.Now().String()
    genesisBlock.Data = "我是创世区块！"
    genesisBlock.Lasthash =
"0000000000000000000000000000000000000000000000000000000000000000"
    genesisBlock.Height = 1
    genesisBlock.Nonce = 0
    genesisBlock.DiffNum = 0
    genesisBlock.getHash()
    fmt.Println(*genesisBlock)
    //将创世区块添加进区块链
    blockchain = append(blockchain, *genesisBlock)
    for i := 0; i < 10; i++ {
        newBlock := mine("天气不错" + strconv.Itoa(i))
        blockchain = append(blockchain, newBlock)
        fmt.Println(newBlock)
    }
}
```

预期结果如下：

练习 1：根据 pow 共识原理，完善代码，形成可运行的 pow 共识模型。

练习 2：pow 的本质就是看谁能先解出一道数学题，同学们可以按照这种思想设计一种 pow 类型的共识算法，并且上述代码风格，使用 golang 实现 pow 类型的共识算法。

实验报告思考题：回答以下问题：

如何调整挖矿难度？

Pow 挖矿对 CPU 要求高还是内存要求高？

你能想出哪些其他的工作量证明的共识实现？

实验 2 PoS 共识的实现

相比于 PoW 共识算法的区块结构，我们增加了挖出块的地址 address 字段，代表着产生本区块的挖矿节点的区块

```
 type block struct {
    //上一个块的 hash
    prehash string
    //本块 hash
    hash string
    //时间戳
    timestamp string
    //区块内容
    data string
    //区块高度
    height int
    //挖出本块的地址
    address string
}
```

其中我们还需要定义挖矿节点的数据结构，

```
 //代表挖矿节点
type node struct {
    //代币数量
    tokens int
    //质押时间
    days int
    //节点地址
    address string
}
```

系统初始化时，需要定义概率池，使得每个挖矿节点挖出矿的概率和持有币的数量和质押时间有关，伪代码如下，需要学生完善。

```
    //初始化
func init() {
    //手动添加两个节点
    mineNodesPool = append(mineNodesPool, node{1000, 1, "AAAAAAAAAA"})
    mineNodesPool = append(mineNodesPool, node{100, 3, "BBBBBBBBBB"})
    //初始化随机节点池（挖矿概率与代币数量和币龄有关）
    for _, v := range mineNodesPool {
        for i := 0; i <= v.tokens*v.days; i++ {
            probabilityNodesPool = append(probabilityNodesPool, v)
        }
        fmt.Println(len(probabilityNodesPool))
    }
}
```

挖矿过程的函数如下

```
    func getMineNodeAddress() string {
}
```

主函数 main.go 如下：

```
    func main() {
    //创建创世区块
    genesisBlock :=
block{"0000000000000000000000000000000000000000000000000000000000000000", "", time.Now().Format("2006-01-02 15:04:05"), "我是创世区块", 1, "0000000000"}
    genesisBlock.getHash()
    //把创世区块添加进区块链
```

```
    blockchain = append(blockchain, genesisBlock)
    fmt.Println(blockchain[0])
    i := 0
    for {
        time.Sleep(time.Second)
        newBlock := generateNewBlock(blockchain[i], "我是区块内容", "00000")
        blockchain = append(blockchain, newBlock)
        fmt.Println(blockchain[i+1])
        i++
    }
}
```

最后结果如下，可以得出挖出矿的概率和挖矿节点持有代币数量和质押时间是正相关的。

练习 1：根据 pos 共识原理，完善代码，形成可运行的 pos 共识模型。

实验报告思考题：回答以下问题：

为什么出块概率需要和质押时间有关？

pos 共识相比 pow 共识具备哪些优点？

实验 3 DPoS 共识的实现

本实验仅供学有余力的同学完成。

一个区块链项目的受托人个数由项目发起方决定，一般是 10 见证人。任何一个持币用户都可以参与到投票和竞选见证人这两个过程中。用户可以随时投票、撤票，每个用户投票的权重和自己的持币量成正比。投票和撤票可以随时进行，在每一轮(round)选举结束后，得票率最高的 10 个用户则成为该项目的见证人，按照随机顺序负责打包区块、维持系统的运转并获得相应的奖励。

DPoS 伪代码：

```
dlist_i = get N witnesses sort by votes //由 100 个投票人从 10 个候选人中选择 3 个挖矿节点
dlist_i = shuffle(dlist_i) //随机改变挖矿顺序
for i range dlist_i
    generate block by dlist_i; //挖矿节点依次出块
```

最后达到的运行效果如下：

练习 1：根据 dpos 共识原理，完善代码，形成可运行的 dpos 共识模型。

五、参考文献

[1] https://golang.org，访问时间：2021.10.19

[2] https://code.visualstudio.com，访问时间：2021.10.19

[3] https://www.jetbrains.com/go，访问时间：2021.10.19

[4] https://zh.wikipedia.org/wiki，访问时间：2021.10.19