实验地点： 4b201 学号: 2111605074 姓名: 马驭时 日期： 2023.10.11

|  |  |  |  |
| --- | --- | --- | --- |
| 实验题目：  实验一：PoW共识机制的实现 | | | |
| 实验目的：  1.理解PoW共识算法的背景和基础知识；  2.掌握hash函数的特点和用法；  3.掌握PoW算法的实现方法； | | | |
| 实验原理及内容：  1.比特币PoW共识算法   * 计算Y=SHA256(str,x) * PoW求解：已知str的值，求Y的前D位都是0的数值N? * 例如：Y=SHA256(“hello”, x)，其中D是难度。      1. Go语言中SHA256的用法可参考4-sha256.go,5-sha256.go，6-sha256.go，也可上网查询其详细用法。     ------------------------------------------------------------------------------------------------------------------  任务：使用go语言实现上述挖矿程序。要求给定参数初始值str和难度值 d，求出nounce,图中的x。对于不同的难度值，给出挖矿时间和对应的算力。  如：  starting search......  success with nonce 109995  hash is 0000974c4b9e5b01679ffc522dcff9a67b2dc1cacb7d3f91564bae1f425b6767  elasped time:0.2962 seconds  hash\_power:371338 hashes per second  starting search......  success with nonce 2030350  hash is 000009b3111cb851956ed3ebda92b68074aa462b2ac3c82b469bbb36e4d2cc21  elasped time:5.5562 seconds  hash\_power:365419 hashes per second  starting search......  success with nonce 4953412  hash is 000000e69179c04ffeee4fe9de1b83afeda4025b8fae218e314119b27a1654fc  elasped time:13.4515 seconds  hash\_power:368242 hashes per second | | | |
| 实验步骤及结果：  package main    import (  "crypto/sha256"  "encoding/hex"  "fmt"  "strconv"  "strings"  "time"  )    // 产生Hash  func GenerateHashValue(x string, Nonce int) string {  // 按照比特币的写法，将区块的所有属性拼接后做哈希运算  var hashdata = x + strconv.Itoa(Nonce)    // 算哈希  var sha = sha256.New()  sha.Write([]byte(hashdata))  hashed := sha.Sum(nil)    // EncodeToString就是转化为字符串的函数  return hex.EncodeToString(hashed)  }    // 实现PoW  func pow(x string, Nonce int, diff int) (string, int64, time.Duration) {  var n = Nonce  var totalHashes int64 // 总哈希尝试次数  var startTime = time.Now() // 开始时间    for {  hash := GenerateHashValue(x, n)  fmt.Println(hash)    // 判断哈希值前 diff 位是否都是 0  if strings.HasPrefix(hash, strings.Repeat("0", diff)) {  elapsed := time.Since(startTime)  fmt.Println("挖矿成功！")  fmt.Println("哈希值是:", hash)  fmt.Println("花费的时间:", elapsed)  totalHashes += int64(n) // 将最后一次尝试次数加入总哈希次数  return hash, totalHashes, elapsed  } else {  // 随机值自增  n++  totalHashes++ // 记录每次尝试的哈希次数  }  }  return "", 0, 0 // 如果没有找到满足条件的哈希值，返回空字符串、0次哈希尝试和0的持续时间  }    func main() {  hash, hashes, duration := pow("your\_string", 123, 6)  fmt.Println("挖矿总算力:", hashes) // 总哈希尝试次数代表了挖矿的总算力  fmt.Println("挖矿时间:", duration) // 挖矿所花费的总时间  } | | | |
| 实验小结： | | | |
| 成绩 |  | 评阅老师 |  |