**实验1 区块链系统简单实现**

**【实验介绍】**

本实验将参考比特币中的区块结构，使用Java程序设计语言实现一个简单的区块链系统，以帮助读者更好地理解区块链的概念。本实验所实现的简易区块链系统名为minichain，该系统模拟比特币的挖矿过程，使用一个工作线程进行交易的打包、Merkle树根哈希值的计算以及相应的挖矿过程（随机替换nonce值，计算出满足挖矿难度条件的区块哈希值）。正确补全代码包中相应的功能函数后，运行主程序，minichain将产生新的区块。

作为第一个实验，本实验代码量不大，也比较简单，主要目的是使读者通过代码的编写，从工程的角度理解区块链。minichain代码根据比特币中的区块结构（包含区块头和区块体）进行类的设计，建议读者在开始实验之前，阅读代码中关于区块结构的代码，一是可以参考预先实现的一些功能函数，有助于后续代码工作的开展；二是可以帮助读者更好地理解区块链的实际构成，从而使读者从工程角度理解区块链。

在进行实验操作之前，我们首先了解Merkle树的概念和创建过程以及Merkle树是如何在区块链中发挥作用的。

(1) Merkle树概念

Merkle树，通常也被称作Hash树，是存储哈希值的哈希二叉树。Merkle树的叶子是数据块(例如，文件或者文件的集合)或者数据块的哈希值。非叶节点是其对应子节点哈希值串联字符串的哈希，最上层的根节点也由其叶子节点哈希值串联字符串的哈希构成。

Merkle树的特点是底层任意数据的变化，都会传递到其父节点，最终传递到树的根节点。Merkle树主要用来进行完整性验证，在分布式环境中，只需要验证从多台主机获取的数据的Merkle树根哈希的一致性即可验证数据的完整性。与此同时，如果验证得到Merkle树的根哈希不一致，也可以通过逐层下降，快速定位到数据块发生变动的位置。除了完整性验证，Merkle树还可以用于数据的比对，在分布式环境中，系统同步之前，首先需要对比本地副本和远程副本是否一致，如果不一致，需要找出不一致的地方，同步将从此处开始进行，如果一致，则不需要同步。与传递大量数据对数据进行对比相比，仅传递哈希值用于对比可以提高对比的效率。

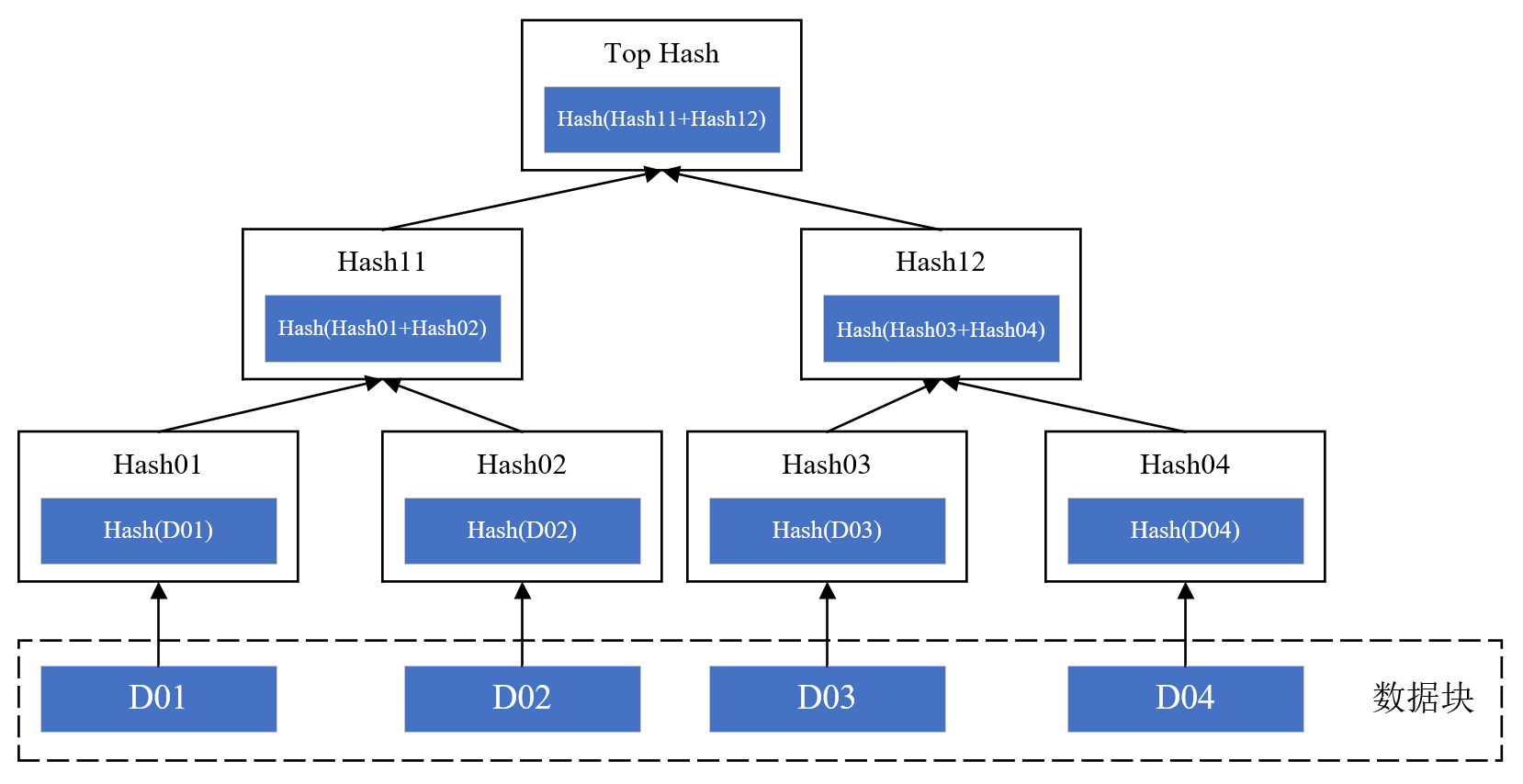


图 1-1 Merkle树结构

(2) 创建Merkle树

在创建Merkle树时，首先要对数据块求哈希，在本实验中我们选用SHA256哈希算法，SHA256可以将任意长的数据块映射到256位的哈希值上。求出数据块的哈希后，将哈希值两两配对，如果是奇数个哈希值，则最后一个哈希值自己与自己配对，计算上一层的哈希，不断地重复进行哈希计算，直到求出Merkle树的根哈希。

假如底层有5个数据块，创建Merkle树的过程如图1-2所示：

第一步：（红色箭头）对数据块做哈希运算，；

第二步：（橙色箭头）相邻两个哈希块串联，然后做哈希运算，；对于，；（蓝色虚线箭头：奇数叶子节点复制自身）

第三步：（黄色箭头）重复第二步；

第四步：（绿色箭头）重复第二步，生成Merkle树根。

创建Merkle树是复杂度（这里指次哈希运算），是数据块的数量。得到Merkle树的树高是。



图 1-2 创建Merkle树

(3) 区块链中的Merkle树

接下来，我们以比特币的数据结构为例来说明区块链中是如何使用Merkle树的。如图1-3所示，数据块由区块头和区块体两部分组成，Merkle树被用于交易的存储，每笔交易生成一个哈希值，在不同交易的哈希值上继续做哈希运算，最终每个区块体中的交易生成唯一的Merkle树根哈希，这个根哈希存储在区块头中。利用Merkle树的特性，如果区块体中的交易1被篡改，将会影响到Hash1，进而影响到Hash12，最终影响到Merkle树的根哈希，由此保证每一笔交易都不被篡改。

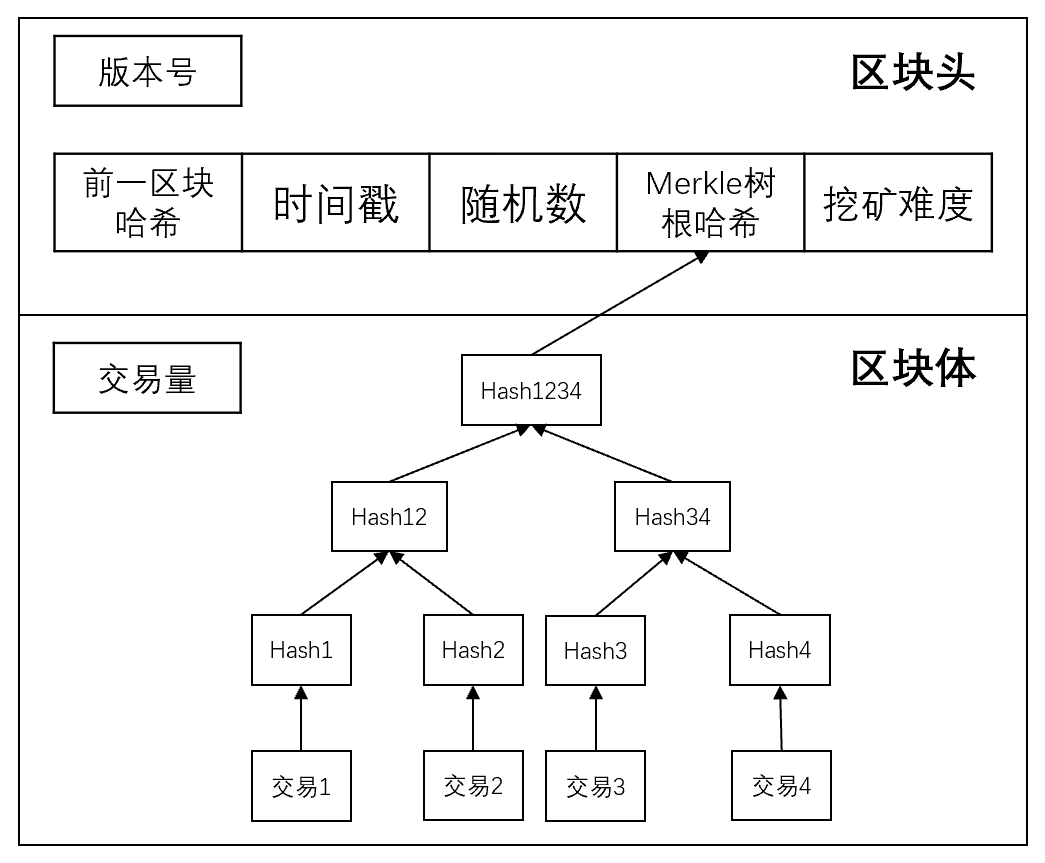


图 1-3 区块的数据结构

以太坊实际上采用MPT（Merkle Patricia Trie，默克尔帕特里夏树）树来保存交易、交易收据和世界状态。MPT树结合了字典树和默克尔树的优点，还根据区块链场景下组织数据的需求进行了一定的改进。这里不做赘述，对MPT树感兴趣的读者可以查阅相关资料，深入了解MPT树，本实验只需要使用Merkle树实现即可。

**【实验要求】**

1. 了解基本区块结构及其必要属性
2. 掌握Merkle树的基本原理，学会计算区块的哈希值
3. 熟悉基于POW共识的挖矿过程

**【实验准备】**

1. **Java环境以及IDE**

Java的安装以及环境配置不在此赘述，有需要的读者可以参考文章[Windows、Mac以及Ubuntu安装和配置JDK的详细过程](https://jiangnanmax.blog.csdn.net/article/details/113941076)，相应的IDE推荐使用IntelliJ IDEA。

1. **项目导入及预览**

使用IDEA打开minichain项目文件夹，等待加载完成，代码主体结构如图1-4所示：

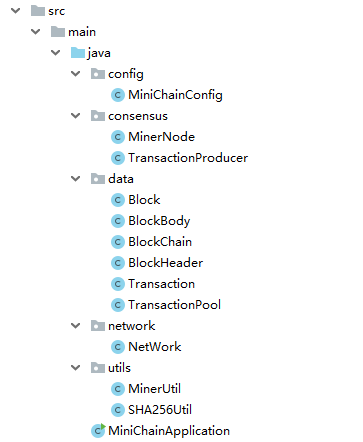


图 1-4 代码结构

这是一个比较简单的demo，代码量并不大，每个类的详细说明以及待实现的功能函数均在代码中进行了相应的注释，这里做一下整体设计思路的说明，读者可以对照代码阅读。

其中，data包中的类是对交易、区块体、区块头以及区块链的抽象，另外，出于代码实现角度，还额外抽象了一个交易池类，该类主要用于存储多笔交易，矿工（挖矿线程）从交易池中获取一批次的交易，进行Merkle树根哈希值的计算和区块体的构造等。

相应地，在consensus包中，对矿工进行了类抽象，MinerNode类继承自Thread类，其功能就是从上述交易池中获取一批次的交易，构造出新的区块体、区块头，组合出一个新的区块，然后使用随机值nonce进行区块哈希值的计算，直至哈希值满足难度条件，即可将该区块加入区块链中（简易模拟了挖矿过程），然后继续生成下一个区块。另外还有一个用于生成随机交易的类，即TransactionProducer类，该类同样继承自Thread类，其功能即随机生成一批次交易放入交易池中，供矿工线程使用。

config包中的MiniChainConfig类中存储了该简易区块链系统的几个关键配置。一个是挖矿的难度设置，即规定了新的区块的哈希值至少以几个0开头才满足难度条件，这里默认设置为4，可以通过修改难度值来感受挖矿的速度变化。当然，这只是一个简单的模拟，但是其效果与真实场景是一致的，该数值设置得越大，挖矿的难度也越大，可以将难度值设置为5或者6进行尝试，能够明显感受到控制台的输出变慢了。另一个则是交易池大小的设置，TransactionProducer线程会随机生成交易放入交易池中，直至交易池中的交易数量达到设置的大小，TransactionProducer线程就会进入等待状态，并唤醒MinerNode线程进行交易打包和后续操作；而MinerNode线程会一次性取出交易池中的所有交易进行交易打包和哈希计算，并在完成其挖矿操作后，进入等待状态，唤醒TransactionProducer线程进行其工作。这其实就是一个最简单的生产者消费者模型，且只有一个生产者线程和消费者线程，如感兴趣，可以尝试增加多个消费者线程，也就是多个矿工线程同时进行挖矿。这里默认将交易池的大小设置为64，不建议对该数值进行修改，设置为2的n次方是为了便于Merkle树根哈希值的计算。

另外，network包中的NetWork类则组合了上述几个功能类，提供两个主要工作线程的启动入口，实现整个系统的运作。utils包则提供了两个可直接调用的工具类，一个是使用SHA256算法进行哈希值计算，另一个则是用于判断区块的哈希值是否满足难度条件。

**【实验过程】**

1. 代码实现

上述内容对minichain区块链系统进行了介绍，相信你已经对该系统有一个比较整体的认识了。接下来就轮到你进行代码实现了，这里已经实现了整体的系统框架，关于类的代码抽象、交易池实现、线程实现以及工具类等均已实现，主要是对MinerNode中的几个功能函数进行了留空，需要你正确补全后系统才能完整的运行起来。

这里需要你进行补全的功能函数分别是:

(1) MinerNode类中的getBlockBody函数，该方法根据传入的参数中的交易，构造并返回一个相应的区块体对象，还需要根据这些交易计算Merkle树的根哈希值；

(2) MinerNode类中的mine函数，在循环中完成"挖矿"操作，其实就是通过不断的变换区块中的nonce字段，直至区块的哈希值满足难度条件，即可将该区块加入区块链中；

(3) MinerNode类中的getBlock函数，该方法供mine方法调用，其功能为根据传入的区块体参数，构造一个区块对象返回，也就是说，你需要构造一个区块头对象，然后用一个区块对象组合区块头和区块体。

代码中也进行了注释，详情见源代码。

**注意：**代码中，针对一个对象（区块、交易等）的哈希值计算，均通过SecurityUtil.sha256Digest(Object.toString())完成。

代码已经在每一个数据结构的类实现中重写了 toString方法，因此涉及对象的SHA256哈希值计算时，参数请使用相应对象的toString方法获取，这样才能顺利通过单元测试。

1. 单元测试及系统运行效果

这里针对getBlockBody函数提供了一个单元测试，主要是该函数的实现涉及了Merkle根哈希值的计算，较为重要，因此在补全该函数之后，可以运行test.java.unit中Test类里的测试方法，通过测试即可。

在补全上述三个功能函数后，即可运行整个系统，运行效果如下图所示：加入的区块的哈希值和挖掘到的区块的哈希值保持一致说明代码正确。

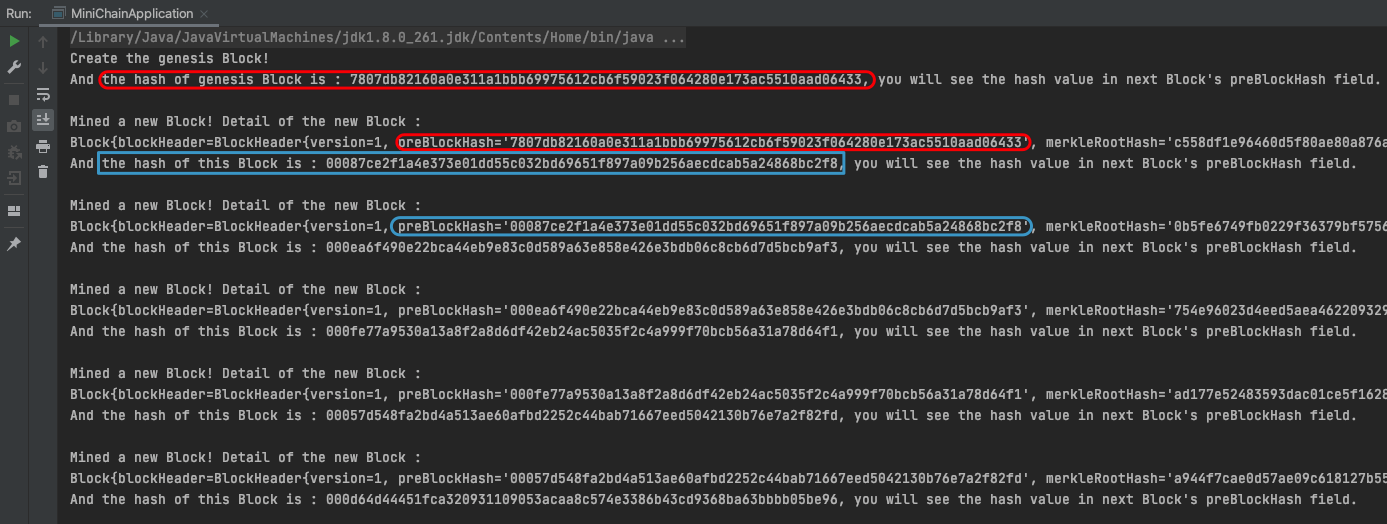


图 1-5 单元测试运行效果

**【实验小结】**

本实验首先从Merkle树入手，介绍了Merkle树的构建过程和Merkle树在区块链中的应用。通过阅读本实验提供的代码，读者将了解区块链和区块的结构，通过编码练习，读者将实现一个可运行的简单区块链系统，理解区块链系统中交易产生、挖矿和区块上链的过程。

本实验中用到的算法都是比较简单的，真实运行的区块链系统中的算法是经过改进的更加高效和安全的，读者可以在完成实验的基础上对实验中提到的算法做更深一步的了解。

**【习题】**

1. 现有一Mekle树如下图所示，请简述树根（Top Hash）的形成过程。

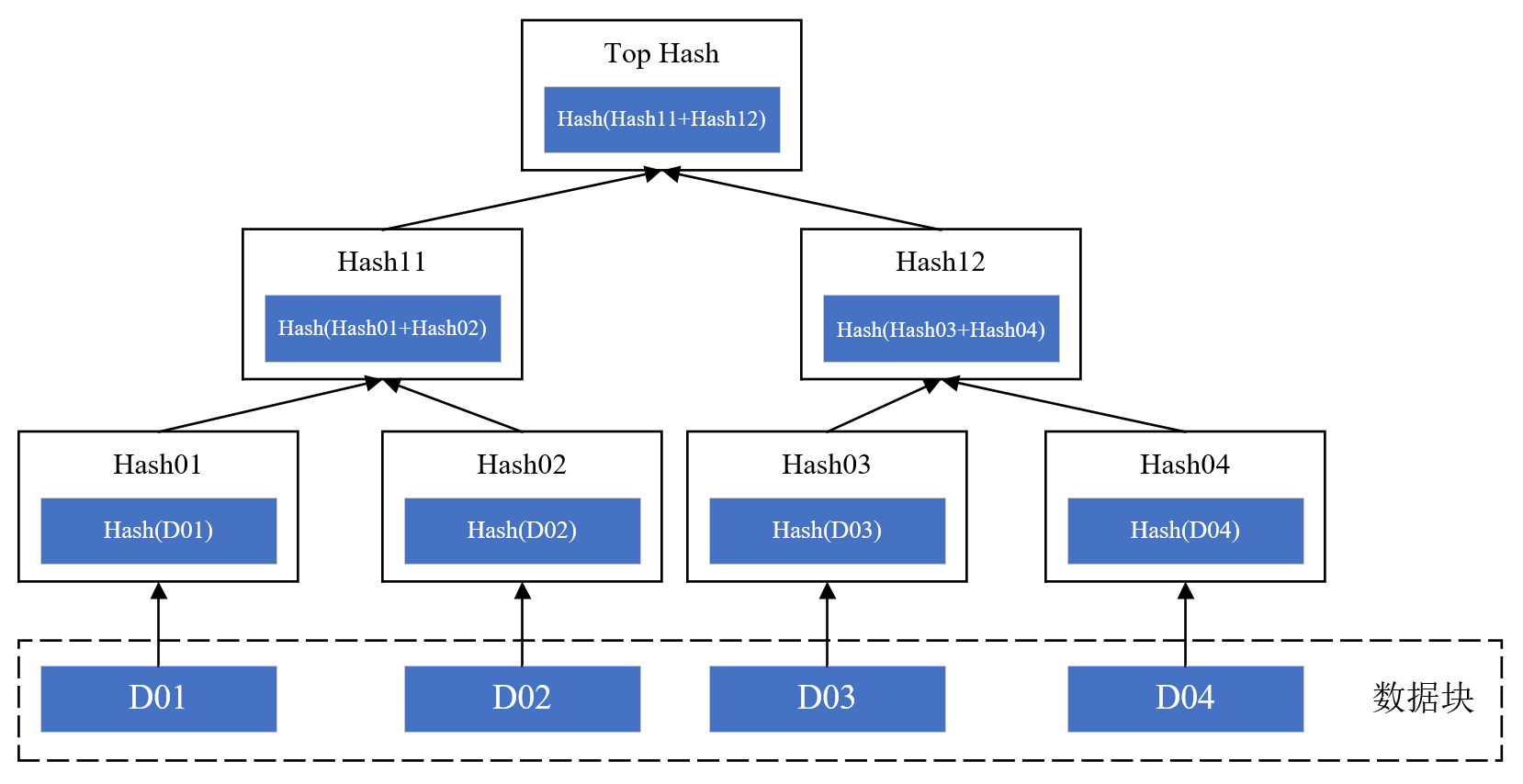


图 1-6 树根形成过程

1. 将数据块 D01、D02、D03、D04 分别进行哈希运算，得到它们的哈希值 Hash01=Hash(D01)、Hash02=Hash(D02)、Hash03=Hash(D03)、Hash04=Hash(D04)。
2. 将相邻的叶子节点的哈希值两两合并，并再次进行哈希运算。这样得到的新的哈希值就是它们的父节点的哈希值。即为Hash11=Hash(Hash01 + Hash02) 和 Hash12=Hash(Hash03 + Hash04)。
3. 将两个中间节点的哈希值再次合并，得到最终的树根（Top Hash）的哈希值。在这个例子中，最终的树根哈希值为Top Hash=Hash(Hash11 + Hash12)。

2. 现有数据A~E，其值分别为A=4，B=3，C=1，D=5，E=8，采用哈希函数hash(x) = x%5，试求由这些数据构成的Merkle树的根哈希。并在图中绘制出Merkle树生成的过程。

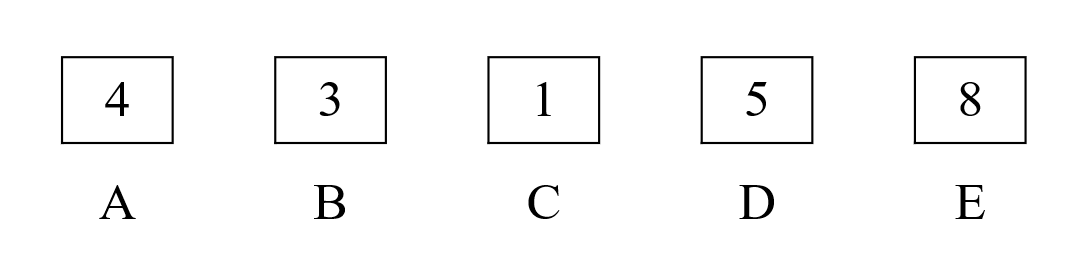


图 1-7 Merkle树原始数据块

3. 简述本实验代码中挖矿的过程。

**【参考文献】**

[1] <https://www.cnblogs.com/fengzhiwu/p/5524324.html>（Merkle Tree学习）

[2] <https://blog.csdn.net/pony_maggie/article/details/74538902>（Merkle树与区块链）

[3] <https://zhuanlan.zhihu.com/p/79978978>（HASH256）

[4] <https://ethereum.org/en/developers/docs/>（以太坊官方文档）