# DPOS算法实验指导书

**一、实验课时**

2课时

**二、实验内容**

理解DPOS算法的工作原理和流程，了解委员会成员的选举和投票过程。

**三、实验环境**

Java开发环境（JDK）：JDK17

IDE工具：vscode

操作系统：windows11

**四、算法描述**

DPOS（Delegated Proof of Stake）是一种共识机制，也是区块链中一种流行的共识算法。与 PoW（Proof of Work）和 PoS（Proof of Stake）不同，DPOS 采用了委托投票的方式来选举出共识节点，由选举出的共识节点来验证和打包交易，生成新的区块。

DPOS 算法的主要思想是，让网络上的持币者通过投票选举出少数几个代表节点，这些节点被授权来验证和打包交易。持币者投票的权重与其持有的货币数量成正比。由于共识节点数量很少，交易验证速度快，处理能力强，同时也保证了网络的分布式特性和安全性。

1. **实验过程**

DPOS算法的流程如下：

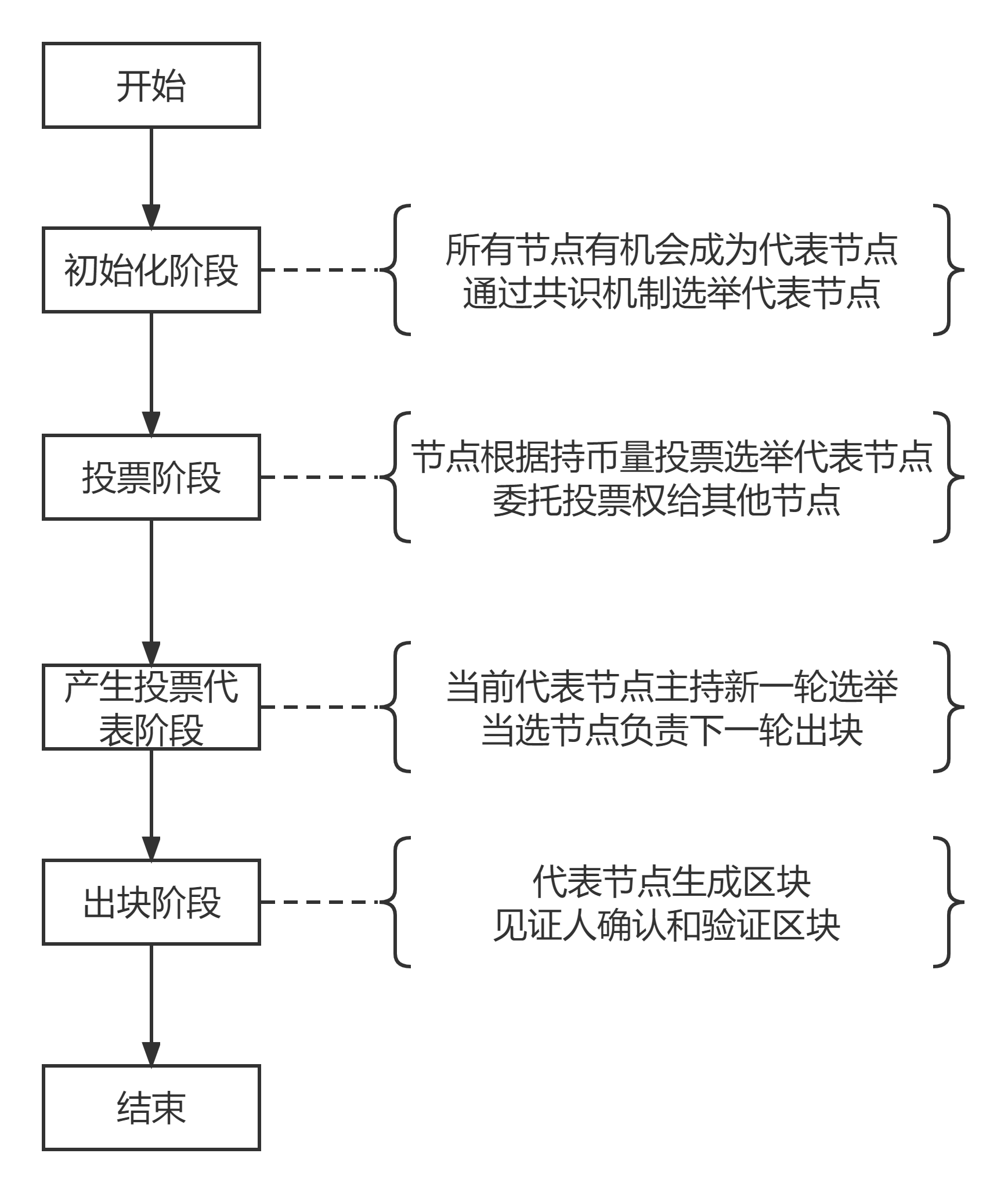


图1：DPos算法流程图

1. 初始化阶段：在区块链网络中，所有的节点都有机会成为“代表”（delegate），即参与出块的节点。在初始阶段，所有节点通过共识机制选举出一组“代表”节点。

区块链类：

public class Blockchain {

// 区块链列表

private static List<Blockchain> *blockchainList* = new ArrayList<>();

// 区块链的难度

private int difficulty;

// 投票列表

private List<Vote> voteList = new ArrayList<>();

// 节点列表

private List<Node> nodeList = new ArrayList<>();

// 区块列表

private List<Block> blockList = new ArrayList<>();

//交易列表

private List<Transaction> transaction = new ArrayList<>();

投票类：

public static class Vote {

private String candidate; // 候选人

private int voteCount; // 获得票数

private int availableVotes; // 节点可用于投票的票数

节点类：

public static class Node {

private String address;

private int availableVotes; // 节点可用于投票的票数

private int voteCount; // 获得票数

private int tokenAmount; // 代币数量

区块类：

public static class Block {

private int index;

private long timestamp;

private List<Transaction> transactionList;

private List<Vote> voteList;

private String previousHash;

private String hash;

private int nonce;

交易类：

public static class Transaction {

private String from;

private String to;

private int amount;

private String signature;

创建创世区块：

public void createGenesisBlock() {

List<Transaction> transactions = new ArrayList<>();

List<Vote> votes = new ArrayList<>();

String previousHash = "";

int nonce = 0;

Block genesisBlock = new Block(0, System.*currentTimeMillis*(), transactions, votes, previousHash, nonce);

blockList.add(genesisBlock);

System.*out*.println("genesisBlock: " + genesisBlock.hash);

}

1. 投票阶段：在下一轮次开始之前，所有节点都可以根据自己的持币量投票选举下一轮的“代表”节点。在此过程中，每个节点可以将自己的投票权委托给其他节点，以提高他们当选的概率。

// 添加节点并随机分配代币数量

List<Blockchain.Node> nodes = new ArrayList<>();

Random random = new Random();

int totalTokens = 10000; // 总代币数量

for (int i = 0; i < 100; i++) {

int tokenAmount = 1 + random.nextInt(totalTokens / 10); // 保证每个节点至少拥有1个代币

totalTokens -= tokenAmount;

String nodeAddress = Blockchain.HashUtil.*sha256*(UUID.*randomUUID*().toString());

Blockchain.Node node = new Blockchain.Node(nodeAddress, tokenAmount);

blockchain.addNode(node);

nodes.add(node);

// 在添加节点的同时，创建对应的投票并添加到投票列表

Blockchain.Vote vote = new Blockchain.Vote(nodeAddress, tokenAmount, tokenAmount);

blockchain.addVote(vote);

System.*out*.println("节点已添加，节点为："+ (i + 1) + ". " + node.getAddress() + "，代币数量为：" + node.getTokenAmount());

}

// 根据分配的代币给予节点票数

for (Blockchain.Node node : nodes) {

int numVotes = node.getTokenAmount(); // 获取节点的代币数量

node.addVote(numVotes); // 给节点增加票数

}

// 进行随机投票模拟

Random random1 = new Random(System.*currentTimeMillis*());

List<Blockchain.Vote> votes = blockchain.getVoteList();

for (Blockchain.Node node : nodes) {

int numVotes = node.getVoteCount(); // 获取节点的票数

for (int i = 0; i < numVotes; i++) {

int candidateIndex = random1.nextInt(votes.size()); // 随机选择候选人索引

Blockchain.Vote vote = votes.get(candidateIndex); // 获取对应的候选人投票

node.vote(vote); // 节点进行投票

}

}

1. 产生投票代表阶段：在每个轮次结束时，由当前的“代表”节点主持新一轮的“代表”节点选举，当选的节点将负责下一轮的出块工作。

// 按票数排序节点

List<Blockchain.Node> sortedNodes = Blockchain.*sortNodesByVoteCount*(nodes);

// 输出票数最高的30个节点

System.*out*.println("票数最高的30个节点：");

for (int i = 0; i < 30 && i < sortedNodes.size(); i++) {

Blockchain.Node node = sortedNodes.get(i);

System.*out*.println((i + 1) + ". " + node.getAddress() + " - 票数：" + node.getVoteCount());

}

1. 出块阶段：在当前的轮次中，由被选中的“代表”节点负责生成区块，而其他节点将成为“见证人”（witness），对该区块的有效性进行确认和验证。

// 创建一个新的区块并添加到区块链

Blockchain.Block newBlock1 = new Blockchain.Block(1, System.*currentTimeMillis*(), new ArrayList<>(), blockchain.getVoteList(), blockchain.getLatestBlock().getHash(), 0);

System.*out*.println("等待添加区块1：");

blockchain.addBlock(newBlock1);

System.*out*.println("区块 1已添加，区块哈希为：" + newBlock1.getHash());

Blockchain.Block newBlock2 = new Blockchain.Block(2, System.*currentTimeMillis*(), new ArrayList<>(), blockchain.getVoteList(), blockchain.getLatestBlock().getHash(), 0);

System.*out*.println("等待添加区块2：");

blockchain.addBlock(newBlock2);

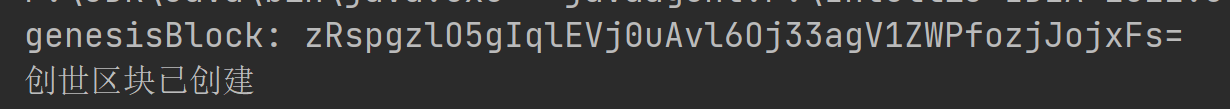
System.*out*.println("区块 2已添加，区块哈希为：" + newBlock2.getHash());

// 验证区块链的合法性

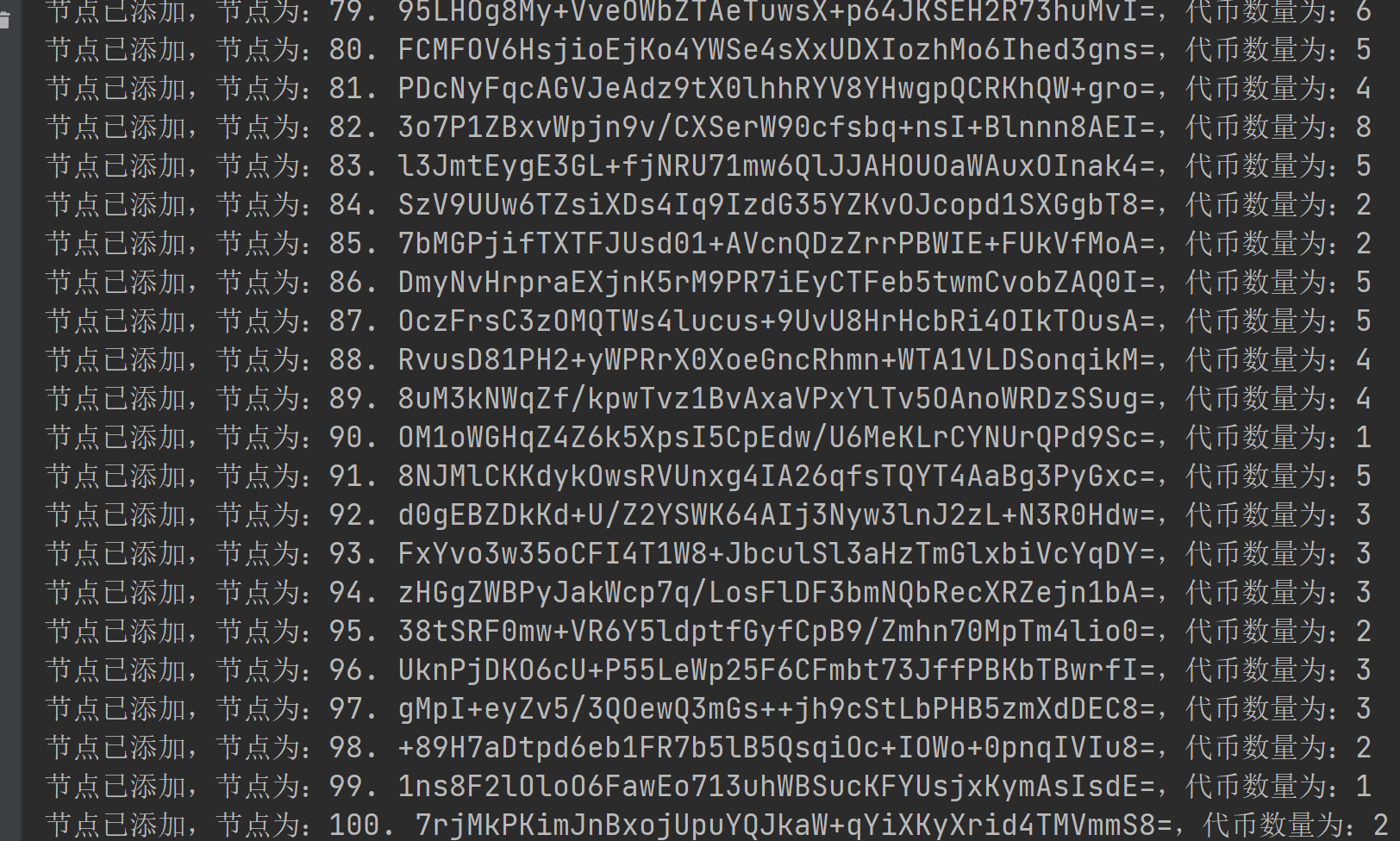
System.*out*.println("区块链的合法性为：" + blockchain.validate());

1. **实验结果**

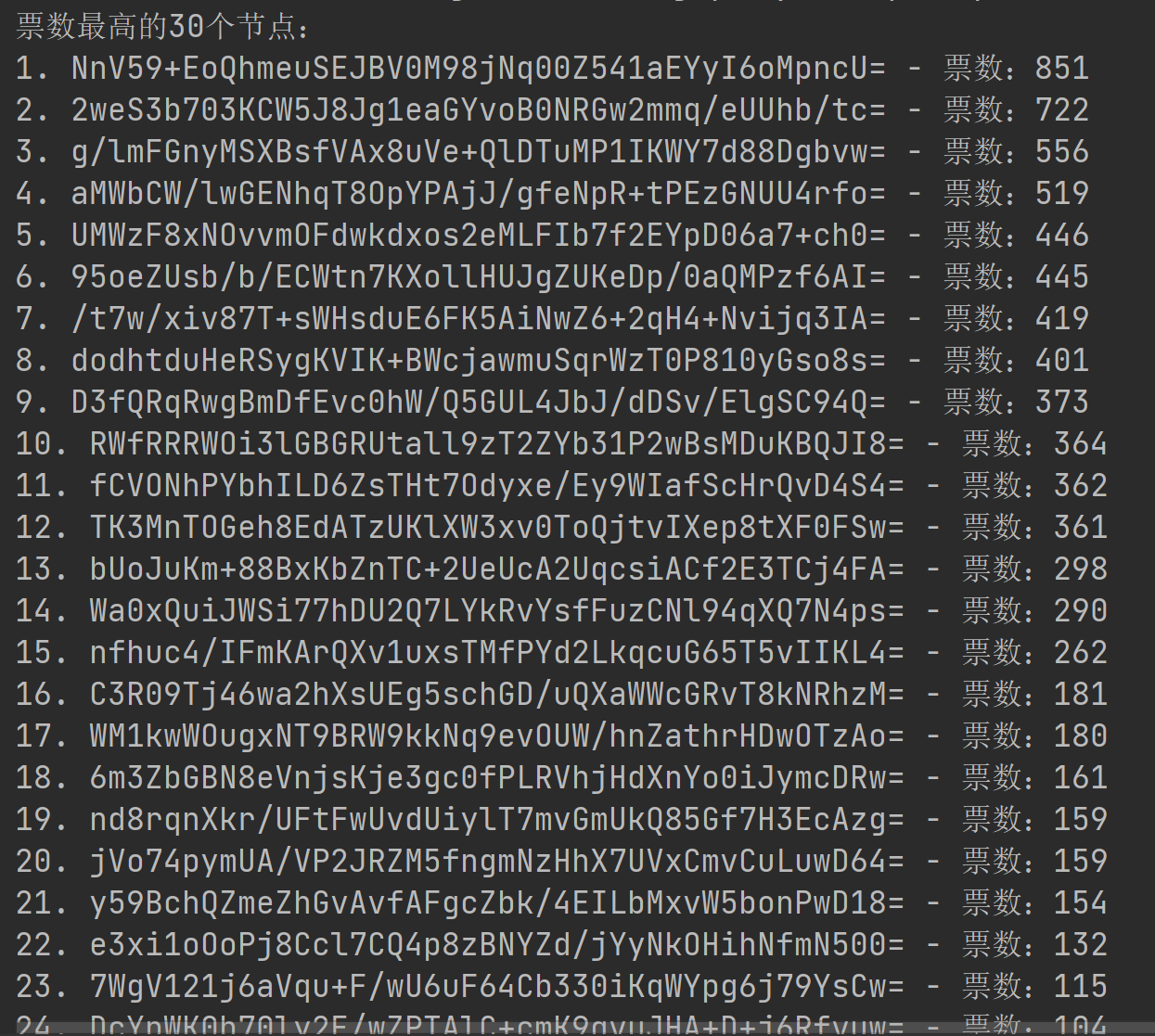
生成创世区块



生成100个节点，并将10000个代币随机分配给这100个节点



进行随机投票并将节点按获得的票数排序，得出排名前30的节点



最后进行出块并验证合法性

