# 实 验 报 告


| | |
|---|---|
| 课程名称 | 密码学基础 |
| 学　　院 | 计算机科学技术学院 |
| 专　　业 | 信息安全 |
| 姓　　名 | 冉津豪 |
| 学　　号 | 17307130179 |

开 课 时 间 **2019** 至 **2020** 学年第 二 学期

| 实验项目名　　称 | Blockchain | 成绩 | |
|---|---|---|---|

## 一、实验目的

1. Understanding basic concepts in blockchain
2. Implementing a blockchain web application

## 二、实验内容

构建一个允许用户通过发布共享信息的应用程序。由于内容将存储在区块链上，所以这些信息将是共识的、不变的、永久的。用户将通过一个简单的 web 界面与应用程序交互。

## 三、实验步骤

我们将通过完成 node_server.py 来完成这个实验。

1. Store transactions into blocks

Block 的初始化将信息保存到一个 Block 对象中。

2. Add digital fingerprints to the blocks

对存有信息的区块进行 sha256 hash 计算，并返回结果。由于 block 存储为 JSON 格式，首先将 JSON 格式转化为字符串，为保证内容顺序设置 dumps 函数参数 sort_keys=True。

```
        def compute_hash(self):
            """
            A function that return the hash of the block contents.
            First converting it into JSON string.
            """
            # WRITE YOUR CODE HERE !
            block_string = json.dumps(self.__dict__,
    sort_keys=True)
            hash = sha256(block_string.encode()).hexdigest()
            return hash
```

3. Chain the blocks

为保证每个区块的完整性，在每一个区块中都将保存前一个区块的 hash 值。同时创建一个初始的区块，将参数归 0，由于该区块中不存在交易信息，所以也不必要计算时间戳。

```
        def create_genesis_block(self):
            """
            A function to generate genesis block and appends it to
    the chain. The block has index 0, previous_hash as 0, and
            a valid hash.
            """
            # WRITE YOUR CODE HERE !
            genesis_block = Block(index=0,
                                  transactions=[],
```

```
                                    timestamp=0,
                                    previous_hash="0")
            genesis_block.hash = genesis_block.compute_hash()
            self.chain.append(genesis_block)
```

4. Implement proof of work (POW) algorithm

为了使计算哈希变得更困难和随机，我们规定每一次记账都需要改变参数 Nonce 使计算出的 Hash 的开头满足 n 个 0。其中 n 就是计算的难度 difficulty。

```
        def proof_of_work(block):
            """
            Function that tries different values of nonce to get a hash that satisfies difficulty criteria.
            """
            # WRITE YOUR CODE HERE !
            block.nonce = 0
            hash = block.compute_hash()
            while not hash.startswith(Blockchain.difficulty * '0'):
                block.nonce += 1
                hash = block.compute_hash()
            return hash
```

5. Add blocks to the chain Mining

在完成上面的步骤之后，我们就可以得到一个符合规定的区块了。现在需要将新构建的区块添加到区块链中。在添加到区块连之前，需要检查区块个区块的 hash 值是否被更改，且是否满足难度要求，区块的 previous_hash 是否和上一个区块的 hash 相同。均满足后，将区块添加到链中。

```
        def is_valid_proof(cls, block, block_hash):
            """
            Check if block_hash is valid hash of block and satisfies
            the difficulty criteria.
            """
            # WRITE YOUR CODE HERE !

            return (block_hash == block.compute_hash()) and block_hash.startswith('0' * Blockchain.difficulty)

        def add_block(self, block, proof):
            """
            A function that adds the block to the chain after verification.
```

```
            Verification includes:
            * Checking if the proof is valid.
            * The previous_hash referred in the block and the hash
        of latest block
              in the chain match.
            """
            # WRITE YOUR CODE HERE !
            # Checking if the proof is valid.
            if not Blockchain.is_valid_proof(block, proof):
                return False
            # The previous_hash referred in the block and the hash
        of latest block
            #          in the chain match.
            previous_hash = self.last_block.hash
            if previous_hash != block.previous_hash:
                return False
            block.hash = proof
            self.chain.append(block)
            return True
```

区块的作用是记录信息，我们将信息的记录，区块的构建工作成为mining，即挖矿。交易信息 一开始将存储在 unconfirmed_transactions 中，挖矿会用这些信息按上述步骤建立一个新区块。并添加到区块链上。

```
        def mine(self):
            """
            This function serves as an interface to add the
        pending
            transactions to the blockchain by adding them to the
        block
            and figuring out Proof Of Work.
            """
            # WRITE YOUR CODE HERE !
            if not self.unconfirmed_transactions:
                return False
            last_block = self.last_block
            new_block = Block(index=last_block.index + 1,

        transactions=self.unconfirmed_transactions,
                              timestamp=time.time(),
                              previous_hash=last_block.hash,)
            proof = self.proof_of_work(new_block)
            self.add_block(new_block, proof)
            self.unconfirmed_transactions = []
            return True
```

6. Establish consensus and decentralization Consensus

区块链并不只是存储在一台计算机上，而是分发到多个节点上。每个检点都可以进行挖矿记录工作。在应用中已经实现新节点的注册接口。为了保证记录信息相同，区块链不产生分支，我们需要实现共识机制：用最长的有效支链作为主链，确定主链后其余支链失效。

为保证有效性，需要先进行整条链的检验。检查整条链的正确性，包括每个区块的 hash 值是否计算正确，该 hash 是否满足难度要求，区块的 previous_hash 是否和上一个区块的 hash 相同。

```
    @classmethod
        def check_chain_validity(cls, chain):
            """
            to check if the entire blockchain is valid.
            """
            # WRITE YOUR CODE HERE !
            previous_hash = "0"
            for block in chain:
                block_hash = block.hash
                if not cls.is_valid_proof(block, block_hash) or previous_hash != block.previous_hash:
                    return False
                previous_hash = block_hash
            return True
```

然后是用共识机制，用最长链替换当前链。consensus()函数在'/mine'路径中被使用，需要构建出新块后在所有节点中判断该节点的链是不是最长。为了获取其它节点的区块链，需要通过 http get 请求，/chain 路径获取区块链。

```
    def consensus():
        """
        A simple consnsus algorithm: If a longer valid chain is found, chain is replaced with it.
        """
        global blockchain
        # WRITE YOUR CODE HERE !
        longest_chain = None
        current_len = len(blockchain.chain)
        for node in peers:
            response = requests.get('{}chain'.format(node))
            length = response.json()['length']
            chain = response.json()['chain']
            if length > current_len and blockchain.check_chain_validity(chain):
                current_len = length
                longest_chain = chain
```

```
16          if longest_chain:
17              blockchain = longest_chain
```

7. Create interfaces

该步骤的代码已经给出，大致说明提供给外部各个接口的作用。

```
@app.route('/new_transaction', methods=['POST'])
def new_transaction():
```

创建新的交易，提供给 app 使用，在新的消息 post 之后发送请求到各个节点。各个节点接收后将交易存储到未确认交易中。

```
@app.route('/chain', methods=['GET'])
def get_chain():
```

查看节点的区块链。

```
@app.route('/mine', methods=['GET'])
def mine_unconfirmed_transactions():
```

节点进行挖矿，即记录交易信息到区块中。

```
@app.route('/pending_tx')
def get_pending_tx():
```

未记录的交易。

8. Build&Run the application

- Running with a single node

1）在 8000 端口运行一个节点。（设置环境变量 Linux 下用 export，windows 中用 set）

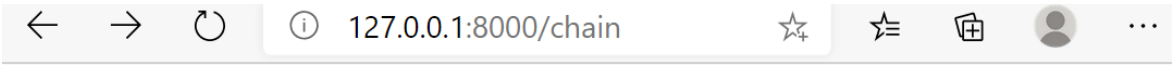
```
set FLASK_APP=node_server.py
flask run --port 8000
```

2）5000 端口运行 app。

```
python run_app.py
```

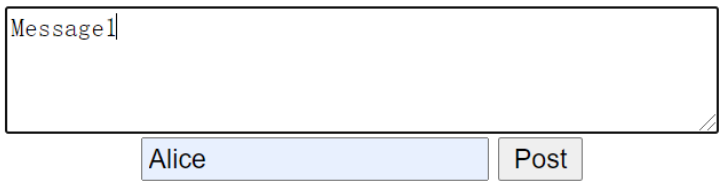
此时即可在 5000 端口使用信息共享服务了。每一个 Post 出的消息需要由节点记录后同步到消息区。

✧ 初始链仅初始区块




{"length": 1, "chain": [{"index": 0, "transactions": [], "timestamp": 0, "previous_hash": "0", "nonce": 0, "hash": "6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffeac292715e5078e631d0c9"}], "peers": []}

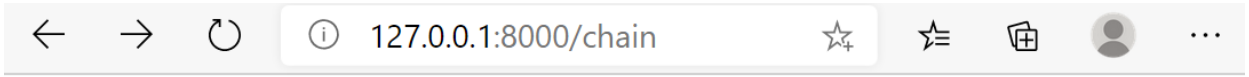
✧ POST 消息



✧ 节点 mine 记录


Block #1 is mined.

✧ 产生新的区块




{"length": 2, "chain": [{"index": 0, "transactions": [], "timestamp": 0,
"previous_hash": "0", "nonce": 0, "hash":
"6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffeac292715e5078e631d0c9"},
{"index": 1, "transactions": [{"author": "Alice", "content": "Message1",
"timestamp": 1592577743.596984}], "timestamp": 1592577756.049935,
"previous_hash":
"6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffeac292715e5078e631d0c9",
"nonce": 129, "hash":
"0046c897a341a2d833dc5ed8ea0de8990bc57b0eff3320a1473a9e0a7346ded1"}],
"peers": []}

✧ 消息同步到消息区

- Running with multiple nodes

  1）分别在 8001, 8002 端口运行节点程序

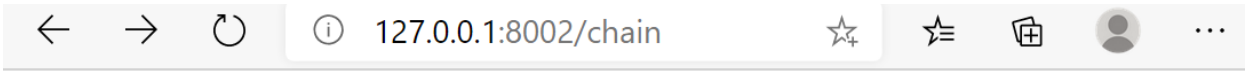
```
flask run --port 8001
flask run --port 8002
```

  2）将两个新节点添加到 8000 同一系统下

```
curl -X POST http://127.0.0.1:8001/register_with -H 'Content-Type:
application/json' -d '{"node_address": "http://127.0.0.1:8000"}'

curl -X POST http://127.0.0.1:8002/register_with -H 'Content-Type:
application/json' -d '{"node_address": "http://127.0.0.1:8000"}'
```

  3）新节点可访问到已存在的最长链。



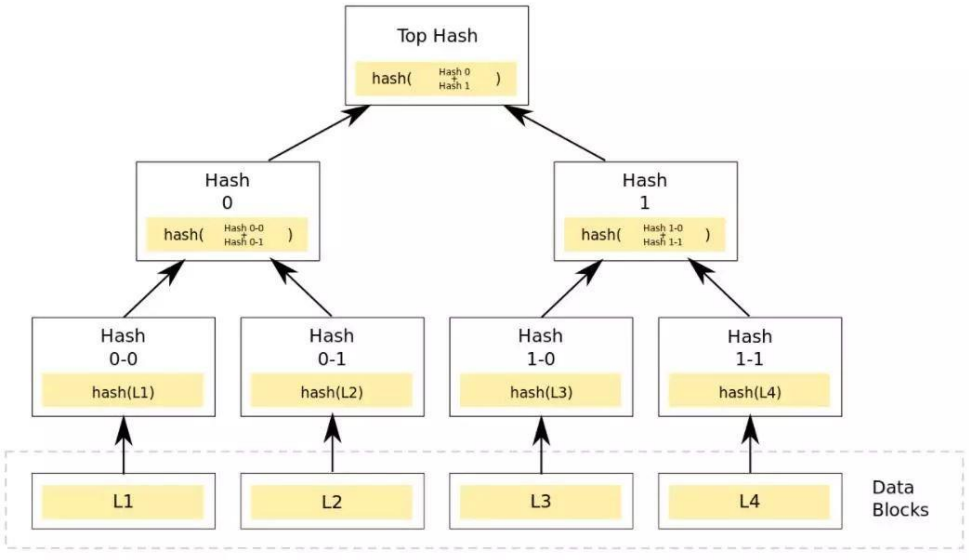

{"length": 4, "chain": [{"index": 0, "transactions": [], "timestamp": 0,
"previous_hash": "0", "nonce": 0, "hash":
"6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffeac292715e5078e631d0c9"},
{"index": 1, "transactions": [{"author": "Alice", "content": "Message1",
"timestamp": 1592577743.596984}], "timestamp": 1592577756.049935,
"previous_hash":

9. Add Merkle tree support

额外的，我尝试添加 Merkle tree 到程序中。

- 计算 Merkle tree 根节点

  我们需要计算出一个节点的 merkle root，用于检验所有交易的完整性。如图所示首先计算所有交易的 hash，然后重复计算合并相邻节点的 hash，知道最后只剩下根节点。返回这个 hash 值。




```
1           def merkle_root(self):
2               length = len(self.unconfirmed_transactions)
3               hashdata = []
4               for i in range(length):
5                   transactions_string = json.dumps(self.__dict__,
    sort_keys=True)
6
    hashdata.append(sha256(transactions_string.encode()).hexdiges
    t())
7               while length > 1:
8                   temp = int(length / 2)
9                   for i in range(temp):
10                      hashdata[i] = sha256((str(hashdata[i * 2]) +
    str(hashdata[i * 2 + 1])).encode()).hexdigest()
11                  if length % 2 != 0:
12                      hashdata[temp] = hashdata[temp * 2]
13                      length = (length + 1) / 2
14                  else:
15                      length = length / 2
16              return hashdata[0]
```

- 添加 Merkler_root 参数

在合适的地方添加 Merkler_root 参数。

首先是 Block 结构的初始化。

```
self.merkle_root = merkle_root
```

接着是创建初始区块 reate_genesis_block。

```
genesis_block = Block(index=0,
                      transactions=[],
                      timestamp=0,
                      previous_hash="0",
                      merkle_root="0")
```

然后是在通过 mine 创建区块时。

```
merkle_root = self.merkle_root()
new_block = Block(index=last_block.index + 1,
          transactions=self.unconfirmed_transactions,
          timestamp=time.time(),
          previous_hash=last_block.hash,
          merkle_root=merkle_root)
```

create_chain_from_dump 是新节点注册时获取已存在的链。

```
block = Block(block_data["index"],
          block_data["transactions"],
          block_data["timestamp"],
          block_data["previous_hash"],
          block_data["merkle_root"],
          block_data["nonce"])
```

verify_and_add_block 是在其余节点成功 mine 之后，添加 block。

```
block = Block(block_data["index"],
          block_data["transactions"],
          block_data["timestamp"],
          block_data["previous_hash"],
          block_data["merkle_root"],
          block_data["nonce"])
```

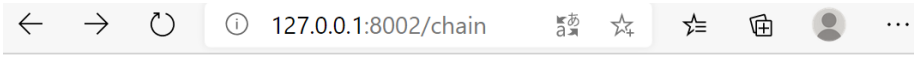
对于区块链的完整性由 hash 来确定，不必再进行 merkle 检验。

添加完成后进行测试。

1） 仅运行一个节点 8000，Post 一个消息，mine 后记录正常。

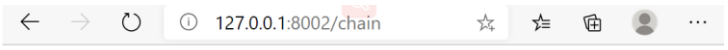

{"length": 2, "chain": [{"index": 0, "transactions": [], "timestamp": 0,
"previous_hash": "0", "merkle_root": "0", "nonce": 0, "hash":
"61226e0edfa757e468d95676f82bdce255348b9973c8d541a3fbf0596a56c487"},
{"index": 1, "transactions": [{"author": "Alice", "content": "Merckle1 test",
"timestamp": 1592581445.5518699}], "timestamp": 1592581446.7396684,
"previous_hash":
"61226e0edfa757e468d95676f82bdce255348b9973c8d541a3fbf0596a56c487",
"merkle_root":
"0c671d68e088a6443c71a7bc0a371e4aabd5ba25c95698e972469988e9c419ff",
"nonce": 643, "hash":
"000fe26d61ca61fdf8d78d4dd598ca461c7703df2a7f88a08297e1f6da689dc3"}],
"peers": ["http://127.0.0.1:8001/", "http://127.0.0.1:8002/"]}

2）运行 8002 节点，尝试注册。注册后同步正常。




{"length": 2, "chain": [{"index": 0, "transactions": [], "timestamp": 0,
"previous_hash": "0", "merkle_root": "0", "nonce": 0, "hash":
"61226e0edfa757e468d95676f82bdce255348b9973c8d541a3fbf0596a56c487"},
{"index": 1, "transactions": [{"author": "Alice", "content": "Merckle1 test",
"timestamp": 1592581445.5518699}], "timestamp": 1592581446.7396684,
"previous_hash":
"61226e0edfa757e468d95676f82bdce255348b9973c8d541a3fbf0596a56c487",
"merkle_root":
"0c671d68e088a6443c71a7bc0a371e4aabd5ba25c95698e972469988e9c419ff",
"nonce": 643, "hash":
"000fe26d61ca61fdf8d78d4dd598ca461c7703df2a7f88a08297e1f6da689dc3"}],
"peers": ["http://127.0.0.1:8002/", "http://127.0.0.1:8001/"]}

3) 新 Post 消息，mine 后同步正常。




{"length": 3, "chain": [{"index": 0, "transactions": [], "timestamp": 0,
"previous_hash": "0", "merkle_root": "0", "nonce": 0, "hash":
"61226e0edfa757e468d95676f82bdce255348b9973c8d541a3fbf0596a56c487"},
{"index": 1, "transactions": [{"author": "Alice", "content": "Merckle1 test",
"timestamp": 1592581445.5518699}], "timestamp": 1592581446.7396684,
"previous_hash":
"61226e0edfa757e468d95676f82bdce255348b9973c8d541a3fbf0596a56c487",
"merkle_root":
"0c671d68e088a6443c71a7bc0a371e4aabd5ba25c95698e972469988e9c419ff",
"nonce": 643, "hash":
"000fe26d61ca61fdf8d78d4dd598ca461c7703df2a7f88a08297e1f6da689dc3"},
{"index": 2, "transactions": [{"author": "Bob", "content": "Merckle2 test",
"timestamp": 1592581790.9202094}], "timestamp": 1592582180.2436776,
"previous_hash":
"000fe26d61ca61fdf8d78d4dd598ca461c7703df2a7f88a08297e1f6da689dc3",
"merkle_root":
"dc5e1720272ba174250be16f9abbe7bece29782e8091dc96e5a5213e1f81d227",
"nonce": 278, "hash":
"0061c20b2f5fb6f0b112b58703151eb4e5c873726f8639c2c6406e3ff127a3a7"}],
"peers": ["http://127.0.0.1:8002/", "http://127.0.0.1:8001/"]}

## 四、实验结果

实验完成，信息系统记录正常。但 app 的实现使得新消息的提交默认只面向 8000，所以仅有 8000 能完成记录。

## 五、实验总结

Blockchain 实验对于区块链的简单复现可以让我们对于区块链的原理、运作方式有一个更直观的认识。需要实现的部分恰到好处，能激发思考，但又不会让人无从下手。

对于实验结果提到的问题，可以对所有节点进行消息提交，这就涉及到在 app 也维护节点地址列表，在新节点注册时，应当向 app 发送请求，在列表中添加地址。为保证公平性，有新消息产生，应按照随机顺序向列表发送信息，使得每个节点接收到未确认交易的时间是公平的。