

# 实验五：区块链

## 一、实验目的

1. 了解区块链内容。
2. 搭建一个自己的区块链。

## 二、实验平台

Server: ubuntu 虚拟机

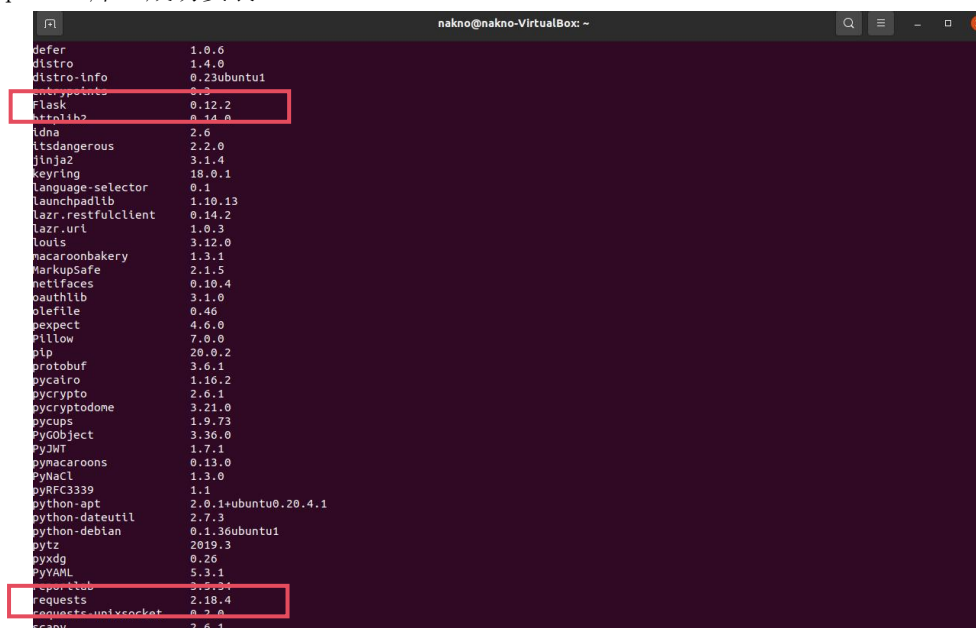
## 三、实验过程及结果分析

步骤一：

在命令行终端中输入 `python3 --version` 命令，查看当前安装的 Python 版本。

```
nakno@nakno-VirtualBox:~$ python3 --version
Python 3.8.10
```

在命令行终端中输入 `pip install Flask==0.12.2 requests==2.18.4` 命令，安装指定版本的 Flask 和 Requests 库，此命令会安装 Flask 版本 0.12.2 和 Requests 版本 2.18.4。安装完成后，在命令行终端中输入 `pip list` 命令，查看已安装的 Python 库列表，确保 Flask 和 Requests 库已成功安装。



```
Flask 0.12.2
requests 2.18.4
```

完成检查后，打开文本编辑器，新建一个名为 `blockchain.py` 的文件。随后编写 `blockchain` 代码。创建一个 `Blockchain` 类，他的构造函数创建了一个初始化的空列表，即用于要存储区块链并且另一个存储交易。

具体代码如下所示。

```

class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []

    def new_block(self):
        # Creates a new Block and adds it to the chain
        pass

    def new_transaction(self):
        # Adds a new transaction to the list of transactions
        pass

    @staticmethod
    def hash(block):
        # Hashes a Block
        pass

    @property
    def last_block(self):
        # Returns the last Block in the chain
        pass

```

下面添加交易到区块的方式。设置 `new_transaction()` 方法用于完成此功能。

```

def new_transaction(self, sender, recipient, amount):
    """
    Creates a new transaction to go into the next mined Block
    :param sender: <str> Address of the Sender
    :param recipient: <str> Address of the Recipient
    :param amount: <int> Amount
    :return: <int> The index of the Block that will hold this transaction
    """

    self.current_transactions.append([
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
    ])

    return self.last_block['index'] + 1

```

在构造函数中创建区块外，还需要补全 `new_block()` 、 `new_transaction()` 和 `hash()` 函数。

```

class Blockchain(object):
    def __init__(self):
        self.current_transactions = []
        self.chain = []

        # 创建创世区块
        self.new_block(previous_hash=1, proof=100)

    def new_block(self, proof, previous_hash=None):
        """
        创建一个新的区块到区块链中
        :param proof: <int> 由工作证明算法生成的证明
        :param previous_hash: (Optional) <str> 前一个区块的 hash 值
        :return: <dict> 新区块
        """

        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

```

随后实现工作量证明，实现一个相似 PoW 算法。

```

def proof_of_work(self, last_proof):
    """
    Simple Proof of Work Algorithm:
    - Find a number p' such that hash(pp') contains leading 4 zeroes, where p is the
    previous p'
    - p is the previous proof, and p' is the new proof
    :param last_proof: <int>
    :return: <int>
    """

    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1

    return proof

    @staticmethod
    def valid_proof(last_proof, proof):
        """
        Validates the Proof: Does hash(last_proof, proof) contain 4 leading zeroes?
        :param last_proof: <int> Previous Proof
        :param proof: <int> Current Proof
        :return: <bool> True if correct, False if not.
        """

        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"

```

然后进行创建节点操作，设置的“Flask 服务器”将扮演区块链网络中的一个节点。添加框架代码如下。

```
# Instantiate our Node
app = Flask(__name__)

# Generate a globally unique address for this node
node_identifier = str(uuid4()).replace('-', '')

# Instantiate the Blockchain
blockchain = Blockchain()

@app.route('/mine', methods=['GET'])
def mine():
    return "We'll mine a new Block"

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    return "We'll add a new transaction"

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

由于已经有了添加交易的方法，所以基于接口来添加交易较为简单，添加事务编写函数代码如下所示。

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # Check that the required fields are in the POST'ed data
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # Create a new Transaction
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201
```

最后添加挖矿代码如下。

```
@app.route('/mine', methods=['GET'])
def mine():
    # We run the proof of work algorithm to get the next proof...
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)

    # We must receive a reward for finding the proof.
    # The sender is "0" to signify that this node has mined a new coin.
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
    )

    # Forge the new Block by adding it to the chain
    previous_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, previous_hash)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
```

## 步骤二：使用 RSA 的加解密

在Postman网站<https://www.postman.com/downloads/>中，安装下载postman Linux x64版本如下。



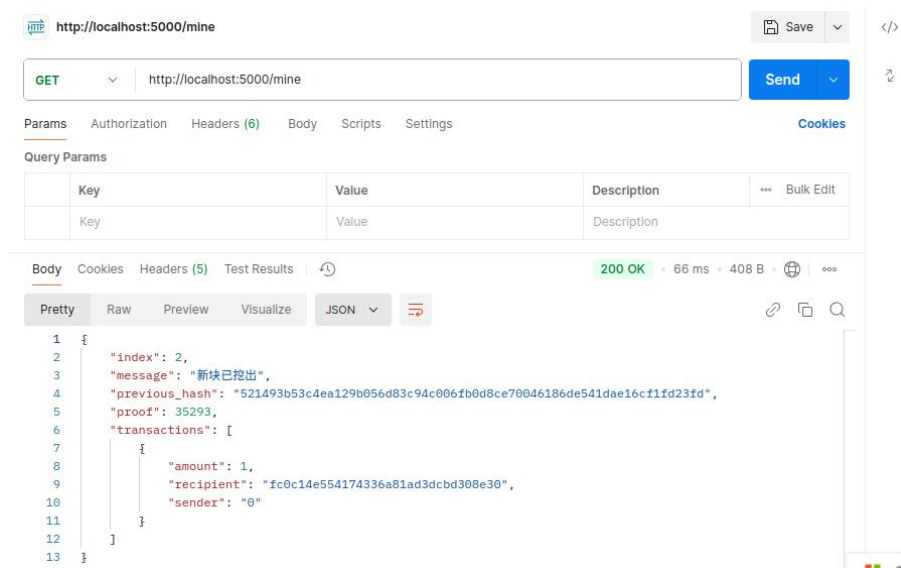
在命令行终端中输入 `sudo chmod 777 postman-linux-x64.tar.gz` 获取权限，后输入 `sudo tar -xzf postman-linux-x64.tar.gz`，解压安装 postman，安装完毕后在终端中输入 `./Postman/Postman`，运行 Postman。



将代码中的 host 从 0.0.0.0 修改成本机 IP 地址 127.0.0.1，随后保存代码。在命令行终端中输入 `python3 blockchain.py`，启动区块链。

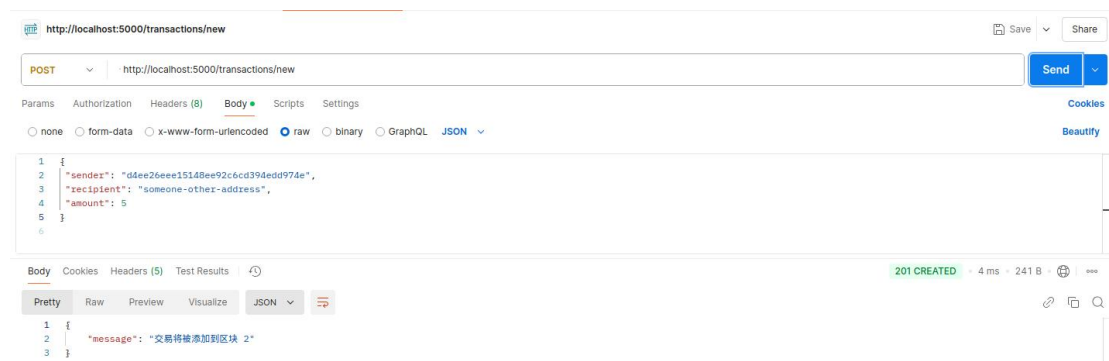
```
nakno@nakno-VirtualBox:~$ sudo python3 blockchain.py
* Serving Flask app 'blockchain'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment
. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

在 Postman 中，通过请求 `http://localhost:5000/mine` (GET) 来进行挖矿如下。



创建一个交易请求，请求 `http://localhost:5000/transactions/new` (POST) 如下图所示。其中 RAW 命令框中，使用以下的 cURL 语句进行设置。

```
{
  "sender": "d4ee26eee15148ee92c6cd394edd974e",
  "recipient": "someone-other-address",
  "amount": 5
}
```



在挖了两次矿后，通过请求 `http://localhost:5000/chain` 可以得到 5000 节点上的块

信息显示如下。

```
1 {
2   "chain": [
3     {
4       "index": 1,
5       "previous_hash": "1",
6       "proof": 100,
7       "timestamp": 1735136960.8522701,
8       "transactions": []
9     },
10    {
11      "index": 2,
12      "previous_hash": "0ea48c3b5805e620a27d821f0334031161895593b80d7de97f7ea5ab728b3091",
13      "proof": 35293,
14      "timestamp": 1735137121.3654861,
15      "transactions": [
16        {
17          "amount": 5,
18          "recipient": "someone-other-address",
19          "sender": "d4ee26eee15148ee92c6cd394ed97de"
20        }
21      ],
22      {
23        "amount": 1,
24        "recipient": "712c68aba2e84a58bcd179780949007d",
25        "sender": "0"
26      }
27    ]
28  },
29  {
30    "index": 3,
31    "previous_hash": "2ee99b6bef8958b0d2f7966c9ad0f4933688d27d193924fc396fee87d09939eb",
32    "proof": 35009,
33    "timestamp": 1735137207.565879,
34    "transactions": [
35      {
36        "amount": 1,
37        "recipient": "712c68aba2e84a58bcd179780949007d",
38        "sender": "0"
39      }
40    ]
41  },
42  "length": 3
43 }
```

此处已完成一个可以接受交易和挖矿的基本区块链。但区块链系统应是分布式的，需要实现一个一致性的算法来使得网络上有多个节点。

首先，新增接口如下：`/nodes/register` 作为接收 URL 形式的新节点列表以及 `/nodes/resolve` 用于执行一致性算法，解决任何冲突，确保节点拥有正确的链。修改 Blockchain 中 `init` 函数并提供一个注册节点方法。同时制定最长的有效链条是最权威的规则。使用此算法，在网络中的节点之间达成共识。设置两个函数，其中第一个函数 `valid_chain()` 通过遍历每个块并验证散列来证明该链是否有效。`resolve_conflicts()` 是一个遍历所有邻居节点的函数，如果找到一个长度大于有效链就取代原链。将两个端点注册到 API 中，一个用于添加相邻节点，另一个用于解决冲突。

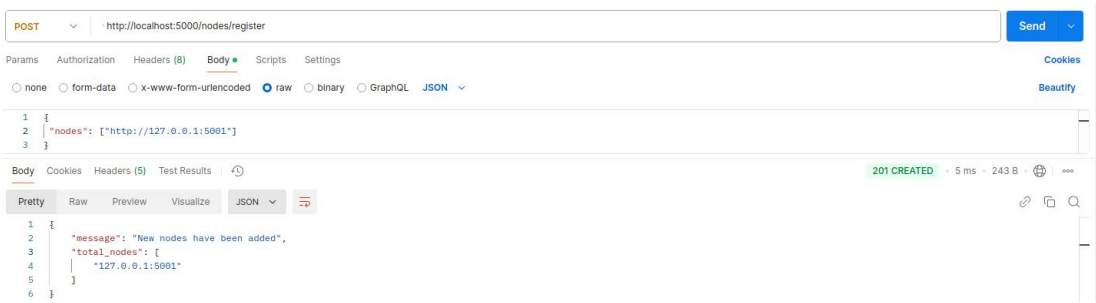
在本地主机上，以不同的端口上创建了另一不同节点，并将其注册到当前节点。在本地主机上的 5000 端口和 5001 端口分别运行 `blockchain.py` 得到如下结果。

```
nakno@nakno-VirtualBox: ~/下载
nakno@nakno-VirtualBox:~/下载$ sudo python3 blockchain.py
[sudo] nakno 的密码:
* Serving Flask app 'blockchain'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:5001
Press CTRL+C to quit

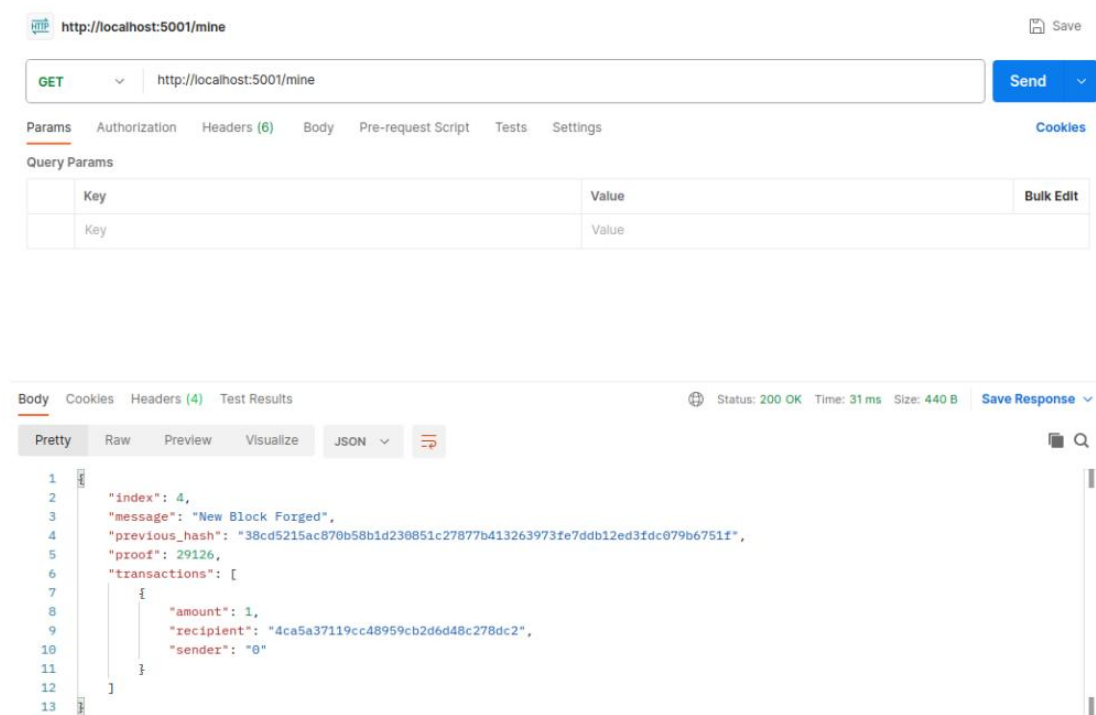
nakno@nakno-VirtualBox:~/下载$ sudo python3 blockchain.py
[sudo] nakno 的密码:
* Serving Flask app 'blockchain'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



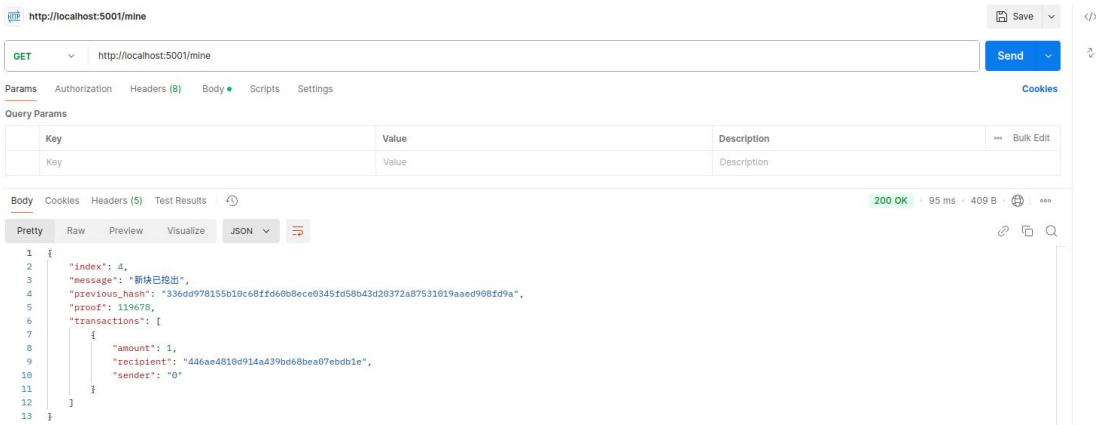
因此，现存在两个节点分别为 `http://localhost:5000` 和 `http://localhost:5001`。注册一个新节点如下。



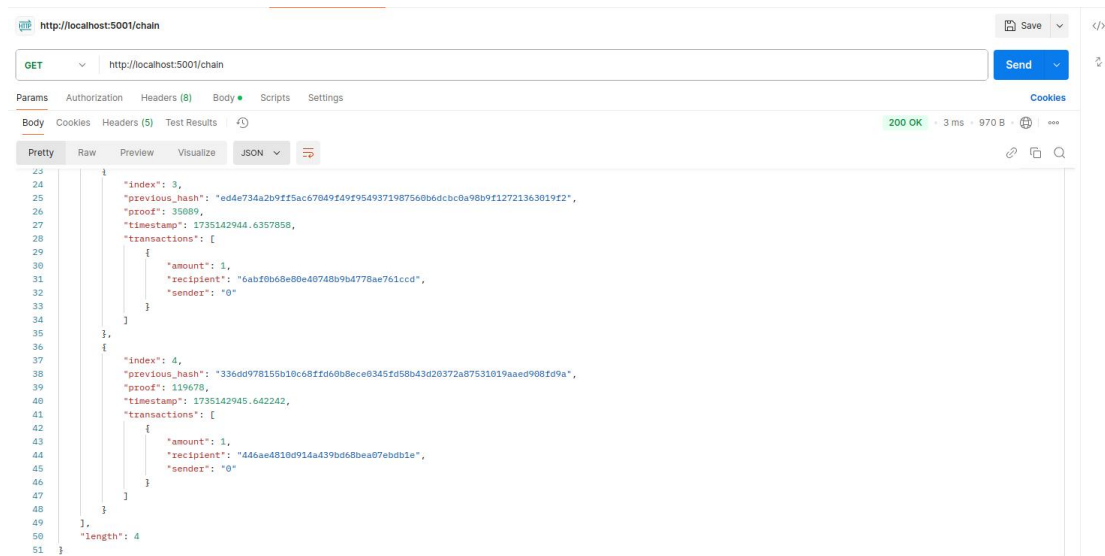
在 Postman 中，通过请求 `http://localhost:5001/mine` (GET)，在 5001 端口上的节点多次挖矿。



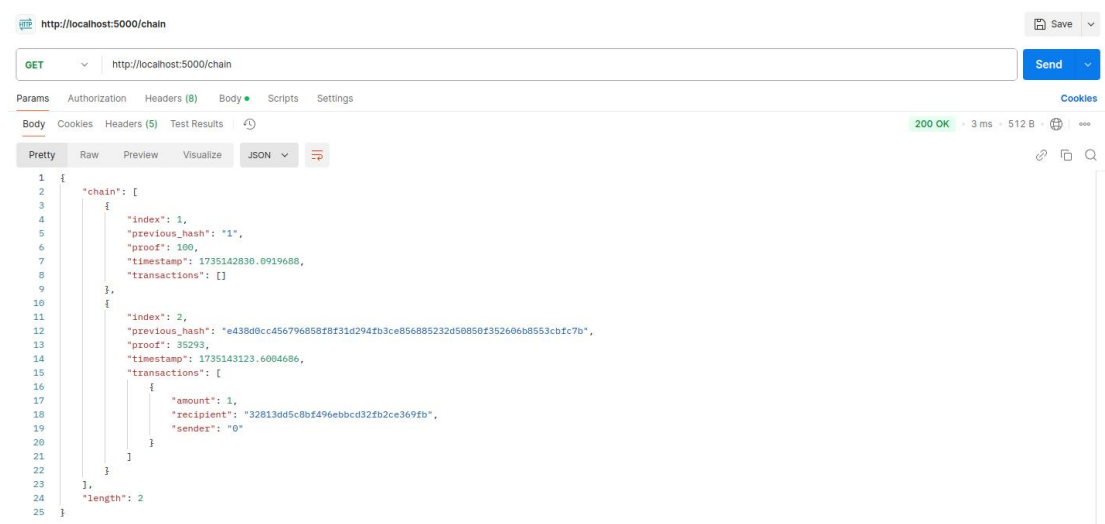
通过请求 `http://localhost:5001/chain` 可以看到 5001 节点的块信息。可以发现，此处 5001 节点上的区块链有 4 个块。



通过请求 `http://localhost:5000/chain`，查看 5000 节点的块信息，发现存在 4 个块。

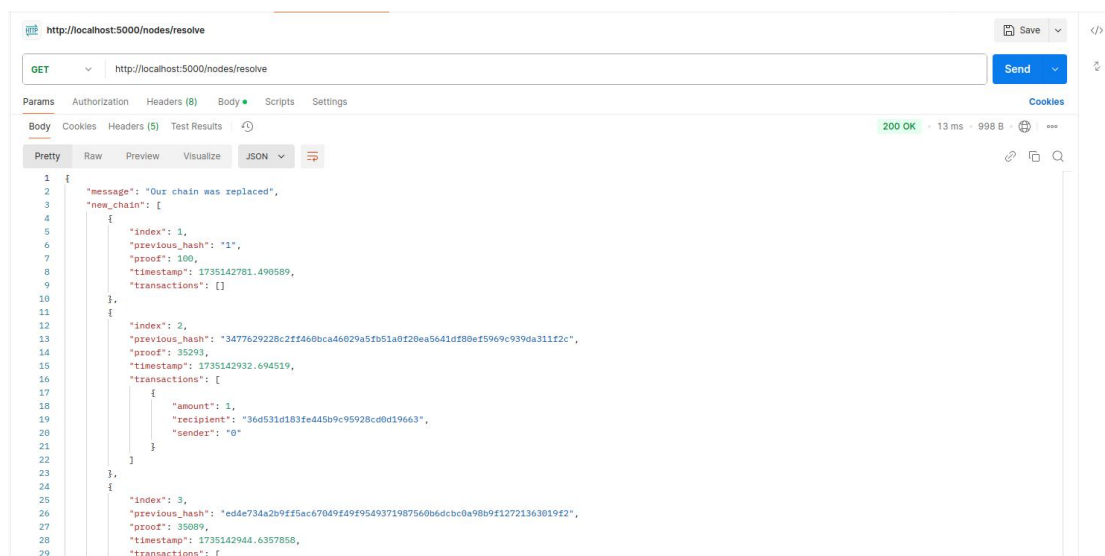


可以看到，此处 5000 节点上的区块链有 2 个块。

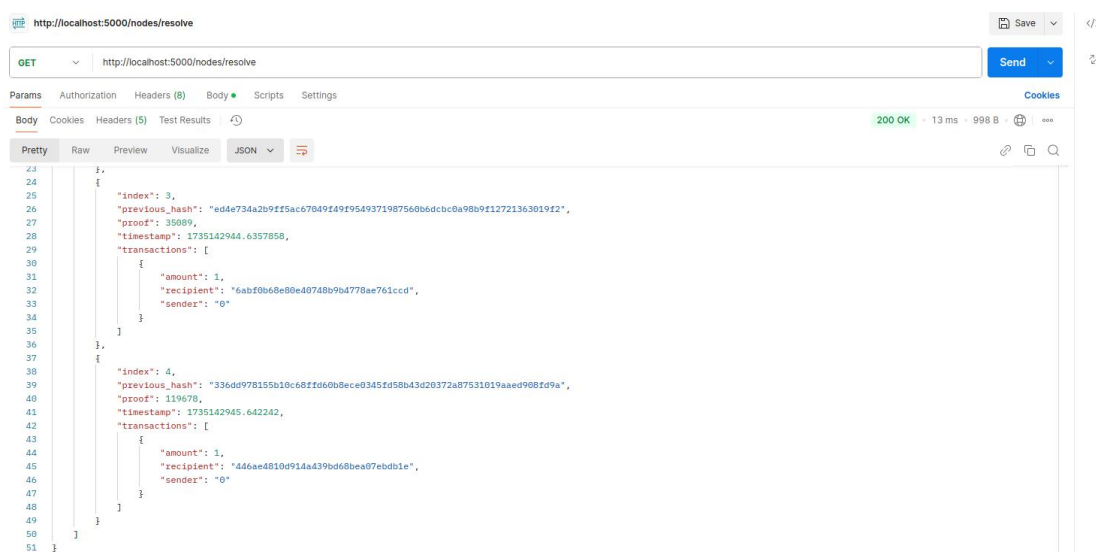


在节点 5001 上挖掘了 4 个块，以确保链条更长。

之后在节点 5000 上调用 GET/nodes/resolve，其中链由一致性算法取代，结果如下。



从图中发现，5000 节点上长度为 2 的链被 5001 节点上长度为 4 的链取代，完成实验。



### 步骤三：相关问题回答

#### 1. 写出一个 block 中 index、timestamp、transactions、proof、previous\_hash 各自的用处。

在区块链中，每个区块包含多个字段，各自发挥不同作用。

index 用于表示当前区块在区块链中的位置或序号。具有区分区块，确保每个区块都有唯一的位置、帮助验证区块的顺序的用处。

timestamp 用于记录该区块被创建的时间。具有提供区块链事件发生的时间信息、检测区块生成间隔是否符合规则，防止某些攻击，如时间戳篡改的作用。

transactions 则用于存储区块中包含的交易信息。它是区块链的核心数据，记录用户之间的资产转移或其他相关信息。具有交易的完整性由数字签名和共识算法确保的重要意义。

proof 表示通过工作量证明机制（PoW）计算得出的合法值。用于证明矿工为生成该区块进行了计算工作，防止篡改。确保只有符合特定条件的区块才能加入链中的作用。

previous\_hash 用于保存前一个区块的加密哈希值、将区块与前一个区块链接起来，形成区块链结构。它确保了区块链的完整性和不可篡改性。篡改任何一个区块都会使链中后续区块的哈希值失效。

这些字段共同确保区块链系统的去中心化、不可篡改性和数据完整性。

#### 2. 这篇文章中的 proof\_of\_work 写的有一点问题，请说明它有什么问题，可能导致什么结果，并进行修正，附在实验报告中。

proof\_of\_work 是一个工作量证明算法。在当前的实现中，proof\_of\_work 使用了一个 while 循环来不断增加 proof，直到 valid\_proof(last\_proof, proof) 返回 True，即哈希值以 4 个零开始。

但是使用了简单的穷举法来寻找合适的 proof 可能会对计算机的性能产生较大压力，随着时间的推移，找到合适的 proof 可能需要进行非常多的尝试。在一个需要高吞吐量的区块链网络中，穷举法会严重影响性能，造成挖矿时间过长，甚至系统拥堵。

同时如果哈希碰巧存在冲突，程序会一直增加 proof，直到找到符合条件的哈希值。在某些情况下，这种方式可能导致程序卡住，特别是当存在某些哈希碰撞或计算能力不足时。

为了提高性能增加限制条件并通过限制最大循环次数，避免算法死循环。在发现无效 proof 时增加日志信息，帮助调试和优化。修正后的代码如下。

```
import hashlib
```



```

class Blockchain:
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.new_block(previous_hash='1', proof=100)
    def proof_of_work(self, last_proof):
        """
        Simple Proof of Work Algorithm:
        - Find a number p' such that hash(pp') contains leading 4 zeroes, where p
is the previous p'
        - p is the previous proof, and p' is the new proof
        :param last_proof: <int> The previous proof
        :return: <int> The new proof
        """
        proof = 0
        max_attempts = 1000000 # Set a reasonable upper limit for attempts
        attempts = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1
            attempts += 1
            if attempts >= max_attempts: # Prevent infinite loops
                raise Exception("Max attempts reached for finding proof of work.")
        return proof
    @staticmethod
    def valid_proof(last_proof, proof):
        """
        Validates the Proof: Does hash(last_proof, proof) contain 4 leading zeroes?
        :param last_proof: <int> Previous Proof
        :param proof: <int> Current Proof
        :return: <bool> True if correct, False if not.
        """
        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"

```

设置了 max\_attempts,即最大尝试次数。如果在这个次数内没有找到符合条件的 proof,程序会抛出异常,避免死循环。并添加了 attempts 计数器来记录尝试次数。如果在规定次数内没有找到有效的 proof,就可以停止计算。这样做能避免程序长时间没有响应,也避免了不必要的性能浪费。当超过最大尝试次数时,抛出异常,保证了系统的稳定性。

### 3. 本次实验为模拟实验,请指出一点未提到的与实际情况不同的地方。

在本次实验中,虽然模拟了一个简单的区块链网络及其工作量证明算法,但与实际的区块链系统相比,仍有一些显著的差异。

在现实中的区块链网络,尤其是像比特币、以太坊等主流区块链,通常存在多个节点并且是去中心化的,网络中每个节点都在不断地同步、验证区块,并使用一致性算法来确保区块链的正确性和一致性。即使使用类似工作量证明或权益证明等机制,这些网络的安全性、

分布式性质、节点间的通信延迟等因素也会极大地影响整个网络的表现和效率。

在本实验中，尽管模拟了多个节点的概念，并通过 `/nodes/resolve` 路由实现了基本的一致性算法，但整个系统仍然比较简单。实验中没有真正考虑到分布式网络的延迟、节点间的网络延迟、不同节点的同步问题等。在实际区块链网络中，节点可能会遇到网络延迟、分叉、节点故障等问题。

因此在实际的区块链系统中，节点间的同步不仅依赖于一个简单的请求响应机制，还需要应对节点之间的延迟和不一致，可能需要更复杂的算法，如 BFT、PBFT 等来保证在恶劣网络环境下也能保持一致。同时，实际系统中，交易不仅要进行验证，还要与其他节点进行广播和同步。每个节点需要判断交易是否有效。

#### 4. 写出你对区块链的理解或者感受（100 字）

区块链不仅改变了金融交易方式，还被广泛应用于供应链管理、版权保护和物联网等领域。作为技术革新，区块链带来了效率与安全的提升，但也面临如扩展性和能耗等挑战，在未来，我相信它可能成为数字社会的重要基石，推动更多行业出现更多的革命性变化。

## 四、实验原理

### 1. 区块链的概念

区块链是一种分布式账本技术，它通过去中心化的方式实现数据的安全性和透明性。每个区块包含了区块链中的一部分数据，而这些区块通过加密的哈希值链接在一起，形成一条不可篡改的链条。区块链的关键特点包括去中心化、数据不可篡改和透明性，适用于需要高度安全性、数据不可篡改和去中介化的场景，如比特币、以太坊等加密货币。

### 2. 区块的组成

区块链中的每个区块通常由以下几个部分组成：

**index**：每个区块的唯一索引，标识区块在区块链中的顺序。

**timestamp**：区块的时间戳，记录区块创建的时间。

**transactions**：区块中所有的交易数据，这些数据在区块链中是公开且不可篡改的。

**proof**：工作量证明通过算法证明区块的计算难度，防止恶意节点的攻击。

**previous\_hash**：前一个区块的哈希值，保证区块链的顺序性和不可篡改性。

### 3. 工作量证明

工作量证明是区块链中的一种共识机制，它要求网络中的节点解决一定的计算难题，从而获得创建新区块的权限。在比特币等区块链网络中，工作量证明确保了区块链的去中心化和安全性。在实验中，`proof_of_work()` 函数是通过不断计算哈希值，直到满足特定条件，例如前四位为零，来找到一个有效的工作量证明。这个过程模拟了区块链网络中矿工挖掘新区块的过程。

### 4. 创建节点和分布式网络

在实验中，使用 Flask 创建了一个简单的区块链节点。每个节点负责处理交易并生成新的区块。通过发送 HTTP 请求，节点可以将交易添加到区块中并进行挖矿。区块链的去中心化特性意味着不同的节点可以通过注册、同步区块链来维持整个网络的共识。

### 5. 区块链中的一致性算法

实验中还涉及了区块链中的一致性算法。当某个节点的区块链长度落后于其他节点时，通过调用 `/nodes/resolve` 接口，可以通过一致性算法将较长的链替换掉较短的链，从而确保整个网络中的区块链一致性。

## 五、实验小结

区块链结构：区块链由一系列按照时间顺序链接的区块组成，每个区块包含交易信息、时间戳、前一区块的哈希值和工作量证明（PoW）等内容。

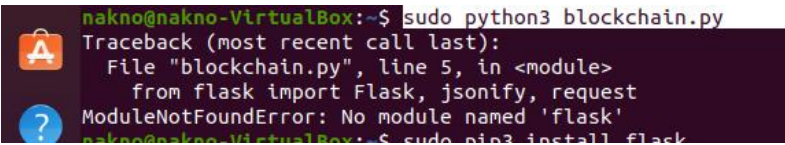
交易和挖矿：通过实现 `new_transaction()` 和 `new_block()` 函数，我掌握了如何将交易添加到区块链中，并通过挖矿算法验证区块的有效性。

分布式节点和共识机制：实验中通过设置多个节点并通过 RESTful API 实现节点间的交互，学习了如何在不同节点之间同步区块链数据。

对区块链的理解：通过这次实验，深入地理解了区块链的去中心化特性及其在金融、供应链等行业中的应用。区块链提供了一种去信任、透明且不可篡改的技术架构，极大地提升了交易的安全性和可靠性。这种技术的广泛应用不仅改变了传统的业务模式，也推动了数字货币如比特币等的发展。

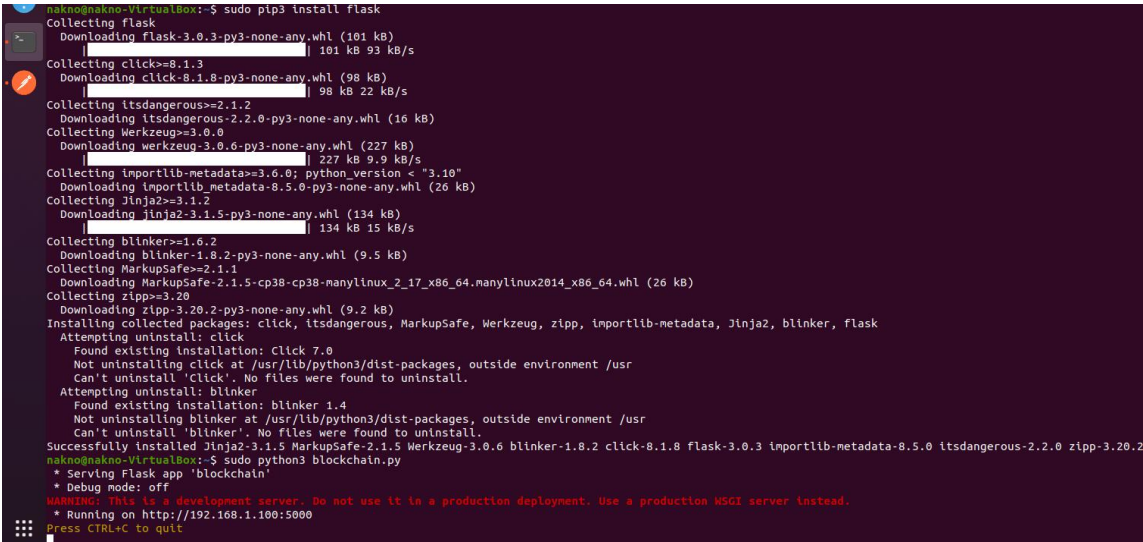
## 六、遇到的问题解决方法

在运行 `blockchain.py` 时，出现 `ModuleNotFoundError: No module named 'flask'` 的报错，但经过检查，该库确实已经安装。



```
nakno@nakno-VirtualBox:~$ sudo python3 blockchain.py
Traceback (most recent call last):
  File "blockchain.py", line 5, in <module>
    from flask import Flask, jsonify, request
ModuleNotFoundError: No module named 'flask'
```

经过查找相关资料，发现 Flask 安装在用户路径 `~/.local/lib/`，而 `sudo` 运行程序时可能无法访问用户路径。因此转为全局安装 Flask，执行 `sudo pip3 install flask` 代码后，成功解决了该问题。



```
nakno@nakno-VirtualBox:~$ sudo pip3 install flask
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
Collecting click>=8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting Werkzeug>=3.0.0
  Downloading Werkzeug-3.0.6-py3-none-any.whl (227 kB)
Collecting importlib-metadata>=3.6.0; python_version < "3.10"
  Downloading importlib_metadata-8.5.0-py3-none-any.whl (26 kB)
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.5-py3-none-any.whl (134 kB)
Collecting blinker>=1.6.2
  Downloading blinker-1.8.2-py3-none-any.whl (9.5 kB)
Collecting MarkupSafe>=2.1.1
  Downloading MarkupSafe-2.1.5-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (26 kB)
Collecting zipp>=3.20
  Downloading zipp-3.20.2-py3-none-any.whl (9.2 kB)
Installing collected packages: click, itsdangerous, MarkupSafe, Werkzeug, zipp, importlib-metadata, Jinja2, blinker, flask
Successfully installed click-8.1.8 flask-3.0.3 importlib-metadata-8.5.0 itsdangerous-2.2.0 Werkzeug-3.0.6 zipp-3.20.2
nakno@nakno-VirtualBox:~$ sudo python3 blockchain.py
* Serving Flask app 'blockchain'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://192.168.1.100:5000
Press CTRL+C to quit
```

## 七、实验心得

通过搭建区块链的简单实现，我深入了解了区块链的基本原理和核心技术。从分布式账本到工作量证明机制，我体验了区块链系统的实际运行。虽然在实验过程中遇到了一些技术

难题，但通过不断调试和相关资料的查询，我克服了这些问题，并顺利完成了实验。通过本次实验，我对区块链技术有了更深入的理解。从创建区块到共识机制，再到实际部署节点，整个过程让我更清楚地理解了区块链如何保证数据的不可篡改性和安全性。在未来，相信这项技术将在更多领域发挥重要作用。

## 八、完整实验代码

```
import hashlib
import json
from time import time
from uuid import uuid4
from flask import Flask, jsonify, request
import requests
from urllib.parse import urlparse

class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.nodes = set() # 存储节点
        # 创建创世区块
        self.new_block(previous_hash='1', proof=100)

    def new_block(self, proof, previous_hash=None):
        """
        创建一个新的区块并添加到区块链中
        :param proof: <int> 工作量证明的值
        :param previous_hash: (Optional) <str> 前一个区块的哈希值
        :return: <dict> 新区块
        """
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

        # 重置当前交易记录
        self.current_transactions = []
        self.chain.append(block)
        return block

    def new_transaction(self, sender, recipient, amount):
```

```

"""
创建一笔交易并加入到下一个区块
:param sender: <str> 发送方地址
:param recipient: <str> 接收方地址
:param amount: <int> 金额
:return: <int> 包含该交易的区块索引
"""

self.current_transactions.append({
    'sender': sender,
    'recipient': recipient,
    'amount': amount,
})

return self.last_block['index'] + 1

@property
def last_block(self):
    # 返回区块链的最后一个区块
    return self.chain[-1]

@staticmethod
def hash(block):
    """
    生成区块的 SHA-256 哈希值
    :param block: <dict> 区块
    :return: <str> 哈希值
    """

    block_string = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_string).hexdigest()

def proof_of_work(self, last_proof):
    """
    简单的工作量证明算法：
    - 找到一个数字 proof，使得 hash(last_proof, proof) 的哈希值以 4 个零
    开头

    :param last_proof: <int> 上一个区块的证明
    :return: <int> 新的证明
    """

    proof = 0
    while not self.valid_proof(last_proof, proof):
        proof += 1

    return proof

@staticmethod
def valid_proof(last_proof, proof):

```

```

"""
验证工作量证明
:param last_proof: <int> 上一个区块的证明
:param proof: <int> 当前的证明
:return: <bool> 验证结果
"""

guess = f' {last_proof} {proof}'.encode()
guess_hash = hashlib.sha256(guess).hexdigest()
return guess_hash[:4] == "0000"

def register_node(self, address):
    """
    将新的节点添加到节点列表中
    :param address: <str> 节点地址, 格式如 'http://192.168.0.5:5000'
    :return: None
    """

    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)

def resolve_conflicts(self):
    """
    一致性算法, 解决区块链冲突
    通过替换本地链为网络中最长的有效链。
    :return: <bool> 如果本地链被替换返回 True, 若没有变化则返回 False
    """

    neighbours = self.nodes
    new_chain = None

    # 获取当前链的长度
    max_length = len(self.chain)

    # 从所有邻居节点获取并验证区块链
    for node in neighbours:
        response = requests.get(f'http://{node}/chain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']

            # 检查是否有更长且有效的链
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain

```



```

        # 如果发现了更长且有效的链，则替换
        if new_chain:
            self.chain = new_chain
            return True

    return False

def valid_chain(self, chain):
    """
    判断给定的区块链是否有效
    :param chain: <list> 区块链
    :return: <bool> 如果有效返回 True，反之返回 False
    """
    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]
        # 验证前一个区块的哈希值是否正确
        if block['previous_hash'] != self.hash(last_block):
            return False

        # 验证工作量证明是否正确
        if not self.valid_proof(last_block['proof'], block['proof']):
            return False

        last_block = block
        current_index += 1

    return True

# Flask 应用
app = Flask(__name__)

blockchain = Blockchain()

@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

```

```

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201

@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    """
    调用一致性算法，解决区块链冲突
    """
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200

@app.route('/mine', methods=['GET'])
def mine():
    # 获取区块链最后一个区块
    last_block = blockchain.last_block
    last_proof = last_block['proof']

    # 找到下一个有效的工作量证明
    proof = blockchain.proof_of_work(last_proof)

    # 奖励矿工，向其账户发送 1 个代币（作为矿工的奖励）
    blockchain.new_transaction(
        sender="0", # "0" 表示奖励矿工
        recipient=str(uuid4()).replace('-', ''), # 使用一个新的随机地址
        amount=1,
    )

```

```

# 创建一个新的区块，并添加到区块链中
previous_hash = blockchain.hash(last_block)
block = blockchain.new_block(proof, previous_hash)

# 返回新创建的区块的信息
response = {
    'message': "新块已挖出",
    'index': block['index'],
    'transactions': block['transactions'],
    'proof': block['proof'],
    'previous_hash': block['previous_hash'],
}
return jsonify(response), 200

@app.route('/chain', methods=['GET'])
def full_chain():
    """
    获取完整的区块链
    """
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5001)

```