

Blockchain

项目复现

首先来复现 github 上的那个项目，项目比较老，4 年前的，好像是国外一位区块链书籍作者写的

我的 python 版本是 3.8，然后安装 flask0.12.2 版本，requests2.18.4 版本

pip install flask==0.12.2

pip install requests==2.18.4

运行，出现了如下报错

```
127.0.0.1 - - [16/Nov/2021 21:36:02] "GET /mine HTTP/1.1" 500 -
[2021-11-16 21:36:03,294] ERROR in app: Exception on /mine [GET]
Traceback (most recent call last):
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/app.py", line 1982, in wsgi_app
    response = self.full_dispatch_request()
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/app.py", line 1614, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/app.py", line 1517, in handle_user_exception
    reraise(exc_type, exc_value, tb)
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/_compat.py", line 33, in reraise
    raise value
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/app.py", line 1612, in full_dispatch_request
    rv = self.dispatch_request()
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/app.py", line 1598, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)
  File "blockchain.py", line 229, in mine
    return jsonify(response), 200
  File "/home/gaoben/.local/lib/python3.8/site-packages/flask/json.py", line 251, in jsonify
    if current_app.config['JSONIFY_PRETTYPRINT_REGULAR'] and not request.is_xhr:
AttributeError: 'Request' object has no attribute 'is_xhr'
```

我怀疑是 flask 库版本的原因，然后更新了 flask 库到最新版 2.0.2

pip install upgrade flask

更新 flask 库之后，运行成功

然后可以开始愉快地运行区块链，用之前定义的五個 API，/mine, /chain,

/transaction/new, /nodes/register, /nodes/resolve 来进行交互，我们主要测试前三个的功能。其中前两个都是 HTTP GET 请求，可以通过浏览器实现。最后一个是 HTTP POST

请求，需要提交表单来交代交易的 sender、recipient、amout 等信息，借助 Postman 实

现比较方便。

本机 IPv4 地址 192.168.1.107

```
wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.107 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::9552:4520:6707:3d58 prefixlen 64 scopeid 0x20<link>
    ether 90:78:41:e8:17:a5 txqueuelen 1000 (以太网)
    RX packets 37649 bytes 23446133 (23.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 31547 bytes 11909989 (11.9 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

gaoben@Panorama39:~$
```

这是通过浏览器进行 HTTP GET 请求获取的信息，可以与 /mine，/chain 交互

GitHub - dvf/blockchain: / × 192.168.1.107:5000/mine × 192.168.1.107:5000/chain

192.168.1.107:5000/mine

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

```
{
  "index": 9,
  "message": "New Block Forged",
  "previous_hash": "23d3d417596daf3d1eed9db47b1cc7ab940253222f6ff751abf5ee524d2b3b83",
  "proof": 19418,
  "transactions": [
    {
      "amount": 1,
      "recipient": "84306d195c4c4cb18717f910789f81c7",
      "sender": "0"
    }
  ]
}
```

GitHub - dvf/blockchain: / × 192.168.1.107:5000/mine × 192.168.1.107:5000/chain × 405 Method Not Allowed × +

192.168.1.107:5000/chain

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

```
{
  "0": {
    "amount": 1,
    "recipient": "84306d195c4c4cb18717f910789f81c7",
    "sender": "0"
  },
  "6": {
    "index": 7,
    "previous_hash": "c96cfa7b6e8b865cfa26e8eb8afc95fa78944d39ffa04436f2deca2027985c6a",
    "proof": 23878,
    "timestamp": 163707086.9275017,
    "transactions": [
      {
        "amount": 1,
        "recipient": "84306d195c4c4cb18717f910789f81c7",
        "sender": "0"
      }
    ]
  },
  "7": {
    "index": 8,
    "previous_hash": "9244f23df6c5653927c968ff76812cf2116c38baa1d8bad5adebe5b32aa1ede",
    "proof": 56879,
    "timestamp": 163707087.4240434,
    "transactions": [
      {
        "amount": 1,
        "recipient": "84306d195c4c4cb18717f910789f81c7",
        "sender": "0"
      }
    ]
  }
}
```

下面演示用 Postman 来进行交互

我们先在 5000 端口的那个节点挖矿 10 次，

Overview GET http://192.168.1.107:5000/mine GET http://192.168.1.107:5000/mine POST http://192.168.1.107:5000/mine + ... No Environment

http://192.168.1.107:5000/mine Save Send

GET http://192.168.1.107:5000/mine Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (4) Test Results Status: 200 OK Time: 51 ms Size: 376 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "index": 10,
3   "message": "New Block Forged",
4   "previous_hash": "4a94e5e114896ba2aa03cebc122f918470b1ebff8518a6568bff78038cc85b98",
5   "proof": 46709,
6   "transactions": [
7     {
8       "amount": 1,
9       "recipient": "3ded6227074d4f55bfd8a47f7c8619c9",
10      "sender": "0"
11    }
12  ]
13 }
```

然后查看区块链，

Overview GET http://192.168.1.107:5000/chain GET http://192.168.1.107:5000/chain POST http://192.168.1.107:5000/chain + ... No Environment

http://192.168.1.107:5000/chain Save Send

GET http://192.168.1.107:5000/chain Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (4) Test Results Status: 200 OK Time: 4 ms Size: 2.28 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "chain": [
3     {
4       "index": 1,
5       "previous_hash": "1",
6       "proof": 100,
7       "timestamp": 1637076480.226395,
8       "transactions": []
9     },
10    {
11      "index": 2,
12      "previous_hash": "c378ca8fb84115d931bfec128802115775cc3ee1dcaba20f6753f81891e51c2",
13      "proof": 20353,
14      "timestamp": 1637076509.2207618,
15      "transactions": [
16        {
17          "amount": 1,
18          "recipient": "3ded6227074d4f55bfd8a47f7c8619c9",
19          "sender": "0"
20        }
21      ]
22    },
23    {
24      "index": 3,
25      "previous_hash": "ad277cd9a8d3a8f3eh78efhh5a03c56e3d67a711d7f7hca2c6fe585a16786c55ea"
```

然后来进行一笔交易，

Overview GET http://192.168.1.10... POST http://192.168.1.107:5000/transactions/new

POST http://192.168.1.107:5000/transactions/new

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "sender": "3ded6227074d4f55bfd8a47f7c8619c9",
3   "recipient": "gaoben",
4   "amount": 3
5 }
```

Body Cookies Headers (4) Test Results Status: 201 CREATED Time: 3 ms Size: 203 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Transaction will be added to Block 11"
3 }
```

现在新加入一个 5001 端口的节点，

Overview POST http://192.168.1.107:5000/nodes/register GET http://192.168.1.107:5000/nodes/register

POST http://192.168.1.107:5000/nodes/register

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nodes": ["http://192.168.1.107:5001"]
3 }
```

Body Cookies Headers (5) Test Results Status: 201 CREATED Time: 5 ms Size: 247 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "New nodes have been added",
3   "total_nodes": [
4     "192.168.1.107:5001"
5   ]
6 }
```

然后在新节点挖矿 8 次，

Overview POST http://192.168.1.10... GET http://192.168.1.10... + ... No Environment

http://192.168.1.107:5000/mine Save

GET http://192.168.1.107:5000/mine Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 31 ms Size: 395 B Save Responses

Pretty Raw Preview Visualize JSON

```
1  {
2    "index": 18,
3    "message": "New Block Forged",
4    "previous_hash": "b9a939e7df295a8ee5cce1c2538890c3d818a434957ac87be4af95e4e0c9243e",
5    "proof": 27177,
6    "transactions": [
7      {
8        "amount": 1,
9        "recipient": "6b446b1f548e4f528b7958915e33b30c",
10       "sender": "0"
11      }
12    ]
13  }
```

最后去旧的节点调用 GET /nodes/resolve, 通过一致性算法获得最长链

Overview GET http://192.168... GET http://192.168... POST http://192.16... GET http://192.168... + ... No Environment

http://192.168.1.107:5000/nodes/resolve Save

GET http://192.168.1.107:5000/nodes/resolve Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 8 ms Size: 4.21 KB Save Responses

Pretty Raw Preview Visualize JSON

```
222  {
223    {
224      "index": 18,
225      "previous_hash": "b9a939e7df295a8ee5cce1c2538890c3d818a434957ac87be4af95e4e0c9243e",
226      "proof": 27177,
227      "timestamp": 1637408599.4518747,
228      "transactions": [
229        {
230          "amount": 1,
231          "recipient": "6b446b1f548e4f528b7958915e33b30c",
232          "sender": "0"
233        }
234      ]
235    },
236    {
237      "message": "Our chain is authoritative"
238    }
239  }
```

这是程序运行时的情况，程序完全正确

```
gaoben@Panorama39:~/blockchain-master$ python3 blockchain.py
* Serving Flask app 'blockchain' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.1.107:5000/ (Press CTRL+C to quit)
192.168.1.107 - - [20/Nov/2021 19:42:42] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:42] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:43] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:44] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:44] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:44] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:45] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:46] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:46] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:49] "GET /chain HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:42:59] "POST /transactions/new HTTP/1.1" 201 -
192.168.1.107 - - [20/Nov/2021 19:43:08] "POST /nodes/register HTTP/1.1" 201 -
192.168.1.107 - - [20/Nov/2021 19:43:13] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:15] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:16] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:16] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:17] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:17] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:18] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:19] "GET /mine HTTP/1.1" 200 -
192.168.1.107 - - [20/Nov/2021 19:43:25] "GET /nodes/resolve HTTP/1.1" 200 -
```

现在开始写实验报告！！！！

1.block 中 index、timestamp、transactions、proof of work、previous hash 各自的用处。

index

用来记录区块的下标

timestamp

Unix 时间戳(Unix timestamp)，或称 Unix 时间(Unix time)、POSIX 时间(POSIX time)，是一种时间表示方式，定义为从格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒（北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒）起至现在的总秒数，不考虑闰秒。

因为标准是统一的，不会跟着时区的改变而改变，所以同一时刻在世界上的不同地点这个数值都是一样的，在计算机中有非常重要的应用。

目前很多计算机系统都是 32 位，使用一个 32 位的带符号数来记录时间戳，能表示的最长时间是 68 年，而实际上到 2038 年 01 月 19 日 03 时 14 分 07 秒，便会到达最大时间，过了这个时间点，所有 32 位操作系统时间便会溢出，导致程序和系统无法工作，这就是著名的 2038 问题。目前，操作系统都在从 32 位向 64 位过渡，64 位带符号数可以记录到约 2900 亿年后的 292,277,026,596 年 12 月 4 日 15:30:08，星期日（UTC）。

说起 32 位和 64 位，我想起实验过程中一个奇妙的经历，就是安装 Postman 的时候，一开始我安装的是 32 位的，结果运行的时候，报错提示缺少依赖库。因为我平时用的都是 Windows 平台，在 Windows 平台 64 位系统是完美兼容 32 位的，只是 64 位的占用内存会更大一点，性能会强一点，这是我对 32 位 64 位的刻板映像。但是在 Linux 平台，32 位程序和 64 位区别还是挺大的，然后我重装了 64 位的 Postman 就能成功运行了。

Unix 系统可以用 `date +%s` 命令来获取当前时间戳

```
Last login: Tue Nov 16 23:46:27 on ttys000
[gaoben@gaobendeMacBook-Air ~ % date +%s
1637078915
```


非常多的编程语言都提供了获取时间戳的 API，其重要性可见一斑

How to get the current epoch time in ...

PHP	<code>time()</code> More PHP
Python	<code>import time; time.time()</code> ↗
Ruby	<code>Time.now</code> (or <code>Time.new</code>). To display the epoch: <code>Time.now.to_i</code>
Perl	<code>time</code> More Perl
Java	<code>long epoch = System.currentTimeMillis()/1000;</code> Returns epoch in seconds.
C#	<code>DateTimeOffset.Now.ToUnixTimeSeconds()</code> (.NET Framework 4.6+/.NET Core), older versions: <code>var epoch = (DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc)).TotalSeconds;</code>
Objective-C	<code>[[NSDate date] timeIntervalSince1970];</code> (returns double) or <code>NSString *currentTimestamp = [NSString stringWithFormat:@"%f", [[NSDate date] timeIntervalSince1970]];</code>
C++11	<code>double now = std::chrono::duration_cast<std::chrono::seconds>(std::chrono::system_clock::now().time_since_epoch()).count();</code>
Lua	<code>epoch = os.time([date])</code>
VBScript/ASP	See the examples
AutoIT	<code>_DateDiff('s', "1970/01/01 00:00:00", _NowCalc())</code>
Delphi	<code>Epoch := DateTimeToUnix(Now);</code> Tested in Delphi 2010.
R	<code>as.numeric(Sys.time())</code>
Erlang/OTP	<code>erlang:system_time(seconds).</code> (version 18+), older versions: <code>calendar:datetime_to_gregorian_seconds(calendar:universal_time())-719528*24*3600.</code>
MySQL	<code>SELECT unix_timestamp(now())</code> More MySQL examples
PostgreSQL	<code>SELECT extract(epoch FROM now());</code>
SQLite	<code>SELECT strftime('%s', 'now');</code>

时间戳（timestamp），一个能表示一份数据在某个特定时间之前已经存在的、完整的、可验证的数据，通常是一个字符序列，唯一地标识某一刻的时间。

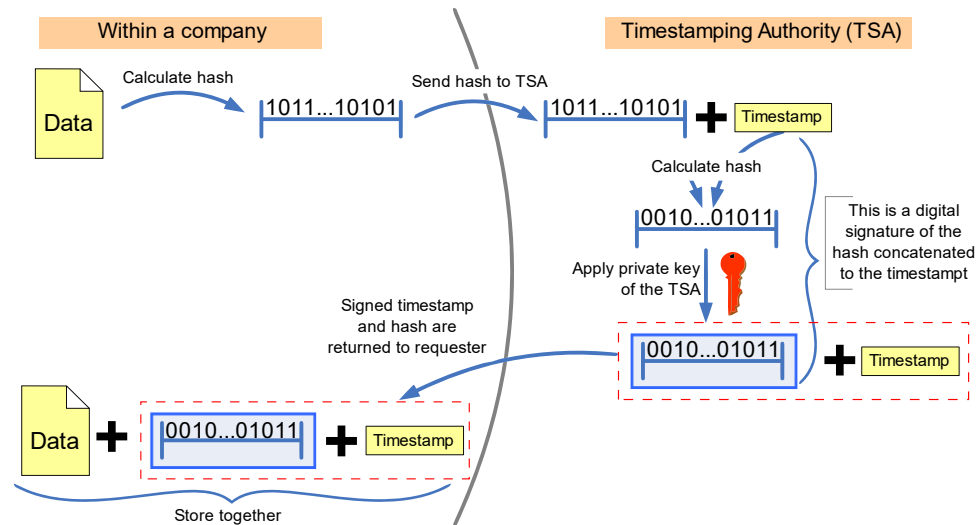
时间戳是使用数字签名技术产生的数据，签名的对象包括了原始文件信息、签名参数、签名时间等信息。时间戳系统用来产生和管理时间戳，对签名对象进行数字签名产生时间戳，以证明原始文件在签名时间之前已经存在。

时间戳的主要目的在于通过一定的技术手段，对数据产生的时间进行认证，从而验证这段数据在产生后是否经过篡改。所以时间戳服务的提供者必须证明服务中使用的时间源是可信的，所提供的时间戳服务是安全的。在实际应用上，它可以使用在包括电子商务、金融活动的各个方面，尤其可以用来支撑公开密钥基础设施的“不可否认”服务。

在区块链中，通过对产生的每一个区块盖上时间戳（时间戳相当于区块链公证人）的方式保证了交易记录的真实性，保证每笔货币被支付后，不能再用于其他支付。在这个过程中，当且仅当包含在区块中的所有交易都是有效的且之前从未存在过的，其他节点才认同该区块的有效性，能一定程度避免二重消费问题。

区块链中如何获取时间戳呢？

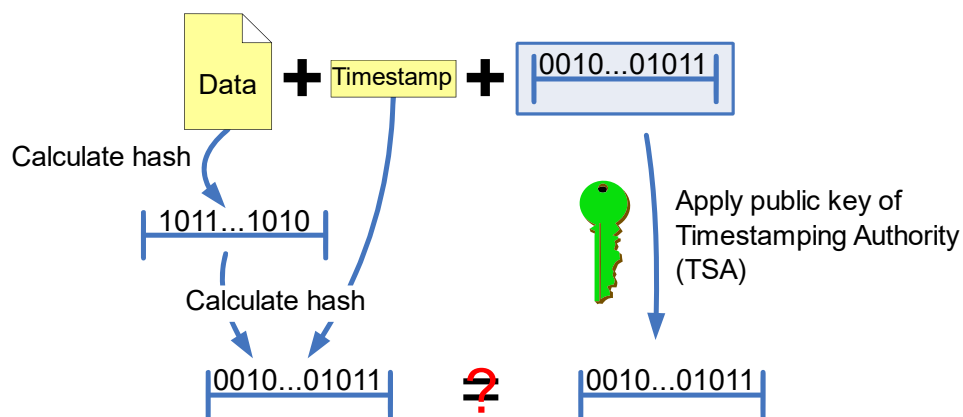
Trusted timestamping



矿工猜对 **Nonce** 之后，得到的 **hash** 值小于等于 **Target Hash**，**hash** 值有个非常形象的比喻，叫做数据的指纹，独一无二。于是将这个 **hash** 值发送给权威机构 **Time Stamping Authority**，简称 **TSA**，**TSA** 获取当前时间戳，将时间戳和发送来的 **hash** 值再次进行 **hash** 运算，得到的结果用自己的私钥加密签名，将加密的结果和使用的时间戳返回给矿工，矿工整合之后就可以将新的区块广播给其他人并将新区块加入区块链。

如何验证时间戳呢？

Checking the trusted timestamp



首先将数据进行 **hash** 运算，得到的 **hash** 值在加上附加的时间戳再次进行 **hash** 运算

将 **TSA** 的签名用公钥解密

比对两个结果是否一致，如果一致，说明数据可信；如果不一致，说明数据很可能有问题，可能被篡改了。

transactions

记录交易记录，每条交易记录包含 3 个关键信息，“sender”、“recipient”、“amount”，也就是交易的发送方、接收方、以及交易的数额。

实际的交易记录非常复杂，Satoshi Nakamoto 中本聪在比特币 9 页的白皮书中有简单的介绍

proof of work

proof of work，工作量证明，这也是理解挖矿原理的关键，让矿工挖矿没那么容易，增加挖矿难度，我认为主要出于以下几点考虑：

- ◆ 让恶意的人有攻击成本，算力消耗、电力消耗等，让他们知难而退，因为他们面对的是区块链网络中所有诚实矿工的算力，进一步提高区块链安全性
- ◆ 工作量证明是解决比特币系统中拜占庭问题的关键，避免有问题的节点（“反叛的将军”）破坏数字货币系统里交易帐的正确性，增加发送信息的成本，降低发送消息的速率，保证在同一时刻只有一个节点或者少数几个节点在进行广播
- ◆ 因为网络延迟和确认时间的因素。由于网络延迟，当矿工开采出一个新区块时，需要广播给世界各地的所有 P2P 节点；全节点得到一个新的区块不是立马就添加到链上面去，还需要对块里面交易进行验证，验证需要花费时间，如果不合法，直接丢弃；如果合法，还需要等待后续的几个区块确认它，大概得等 5 个区块，此时才能放心添加到链上面去；这几个行为都会花费一些时间，因此不能很快的就产生一个新的区块，区块之间得确保一定的时间间隔，比如比特币大概每 10 分钟产生一个新区块
- ◆ “最长链共识”，每次算出来都需要一定的代价，能算出来的算力大概率比较强，更加可信，增强一致性
- ◆ 避免出现分叉 fork，如果很容易就能算出来的话的话，同时很多人算出来，链分叉现象严重，系统的总算力分散到各个分叉链中，这时系统的安全性大幅度降低，黑客可以集中算力进行分叉攻击

那么该如何控制 proof of work 呢？

在 bitcoin 中，区块头部字段中有一个 **Nonce**, **number used only once**，同时还有一个 **Bits** 字段，因为 P2P 网络是在不断变化的，随时有矿工加入和退出，还有硬件性能的提升，系统算力不断变化，为了确保总是平均需要 10 分钟来挖掘一个区块，**Bits** 字段得动态调整，定期调整难度。在每添加 2016 个区块后，区块系统会查看创建这些 2016 个区块所花的时间。正常情况下，需要 20160 分钟，大概 2 周左右。如果超过两周，难度就会降低，如果不到两周，难度就会增加。难度的调整，与交易的数量和金额无关，是在每个完整节点中独立自动发生的，通过调整区块头中的 **Bits** 字段来实现的，需要注意的是，区块头中并没有直接存储全网难度（**difficulty**）的字段，**difficulty** 只是为了让我们直观感受到当前难度的根据 **Bits** 字段算出来的一个数

$$New\ Difficulty = Old\ Difficulty \times (Actual\ Time\ of\ Last\ 2016\ Blocks / 20160\ minutes)$$

大小	字段	描述
4 字节	Version	版本号，用于跟踪软件或协议的升级
32 字节	Hash Previous Block	引用区块链中父区块的哈希值
32 字节	Hash Merkle Root	该区块中交易的 Merkel 树根的哈希值
4 字节	Time	Unix 时间戳，记录当前区块生成的时刻
4 字节	Bits	难度目标，该区块工作量证明算法的难度目标
4 字节	Nonce	用于工作量证明算法的计数器

比特币区块头结构

Bits 有 32 位，存储的是系数/指数格式，前 8 位数字为幂 **exponent**，后 24 位数字为系数 **coefficient**

$$target\ hash = coefficient \times 256^{(exponent - 3)}$$


为了使区块头的 **SHA256** 结果小于某个目标值（**target**），平均要尝试的计算次数，定义为难度（**difficulty**）， $1\ difficulty \approx 2^{32}\ 次 = 4294967296\ 次 \approx 4.2 \times 10^9\ 次 \approx 4G\ 次$ 运算

$$出块时间(单位：秒) \approx difficulty_当前 \times 2^{32} / 全网算力$$

Target Hash 开头几位都是 0，如果计算出来头部的 hash 值小于等于 Target Hash，就认为是算对了，获得记账的权利，并获得相应的区块奖励以及交易手续费，矿工只能从 0 开始调整 Nonce 的值使得整个头部的 hash 值合法。

下面这个是比特币创世区块的一些信息

这是从网站上截取的，他已经帮我们进行了从小端结尾到大端结尾的转换，时区的转换，以及一些特殊字段的生成比如 difficulty

Hash	000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f 
Confirmations	710,086
Timestamp	2009-01-04 02:15
Height	0
Miner	Unknown
Number of Transactions	1
Difficulty	1.00
Merkle root	4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
Version	0x1
Bits	486,604,799
Weight	1,140 WU
Size	285 bytes
Nonce	2,083,236,893
Transaction Volume	0.00000000 BTC
Block Reward	50.00000000 BTC
Fee Reward	0.00000000 BTC

前面提了这么多 proof of work 的优点，那它有什么弊端吗？

- 不少人诟病这种方式浪费电力资源、算力资源，不环保，也有其他的替代方式，比如 proof of stake, proof of burn
- 更重要的一点是限制了交易处理的速度，导致每秒钟最多进行的交易数量有限，不能很高效地流通。比如比特币每秒最多进行 7 次交易，相比之下，Visa 卡每秒可完成 2.4 万次交易；支付宝在双十一这样高峰的日子，每秒必须进行 30 多万次交易

previous hash

后面的区块通过引用前一块的 **hash** 值，将每个独立的区块联系起来，每次产生新的区块都能使前面的区块安全性进一步加强

我们来看看区块链三个著名的问题：

Double-spend

当一笔交易被广播到区块链网络之后，接收到交易的全节点以及挖矿节点会对交易进行验证，检查其是否被花费过，即是否存在于 **UTXO** 中。如果交易输出已不存在于未花费交易列表中，则验证失败，验证失败直接丢弃，不继续广播给其他节点。

另一方面，为了防止一个 **UTXO** 被重复使用的情况，比特币网络中还引入了时间戳的概念。假设用户 **A** 将被认证为 **UTXO** 的 **1 BTC** 同时转账给 **B1**、**B2**，两笔交易仅有一笔会成功完成，因为挖矿节点会选择性的记录优先接收到的或交付手续费更高的那笔交易。当交易被挖矿节点先后记录，根据时间戳的数据，最先被记录的交易才能成功验证。

即使两个挖矿节点分别记录并验证“从 **A** 到 **B1**”以及“从 **A** 到 **B2**”为有效交易，且将各自挖出包含相关交易的新区块同时广播到比特币网络中；双花现象也不会产生。根据比特币协议，当两个节点同时生成新区块时，区块链会出现分叉；只有最先生成新区块并成为当前最长链上的交易，才能被认证。通常有超过六个区块对交易进行确认之后，该转账过程才算成功。

几乎不可能发生二重消费问题，除非某个人掌握了 **51%** 的算力，此时他必然是这个区块链最大的受益者，如果他来二重消费，那么该区块链的价值将大打折扣，届时他就会成为最大的受害者，得不偿失，除非他是个疯子。

51% attack

所谓 **51%** 的攻击，就是利用比特币网络采用 **PoW** 竞争记账权和“最长链共识”的特点，使用算力优势生成一条更长的链“回滚”已经发生的“交易行为”。**51%** 是指算力占全网算力的

51%，比特币网络需要通过哈希碰撞来匹配随机数从而获得记账权，算力衡量的是一台计算机每秒钟能进行哈希碰撞的次数。算力越高，意味着每秒钟能进行多次的哈希碰撞，即获得记账权的几率越高。在理论上，如果掌握了 50%以上的算力，就拥有了获得记账权的绝对优势，可以更快地生成区块，也拥有了篡改区块链数据的权利。实际上，当恶意攻击者持有比特币全网占比比较高的算力时，即使尚未达到 51%的比例，也可以制造相应的攻击，比较典型的就是二重消费，记账后的二重消费。

若掌握了 51%的算力，除了可以修改自己的交易记录外，还可以阻止区块确认部分交易，以及阻止部分矿工获得有效的记账权。但是，拥有 51%的算力也不是万能的，无法修改其他人的交易记录，也不能阻止交易的发出，更不能凭空产生 BTC。

拜占庭将军问题

是由莱斯利·兰伯特提出的点对点通信中的基本问题，含义是在存在消息丢失的不可靠信道上试图通过消息传递的方式达到一致性是不可能的

拜占庭帝国(Byzantine Empire)军队的几个师驻扎在敌城外，每个师都由各自的将军指挥。将军们只能通过信使相互沟通。在观察敌情之后，他们必须制定一个共同的行动计划，如进攻(Attack)或者撤退(Retreat)，且只有当半数以上的将军共同发起进攻时才能取得胜利。然而，其中一些将军可能是叛徒，试图阻止忠诚的将军达成一致的行动计划。更糟糕的是，负责消息传递的信使也可能是叛徒，他们可能篡改或伪造消息，也可能使得消息丢失。

2.修改博客中的 proof of work

因为 proof 只和 last proof 有关，先利用 last proof 算出 proof，然后当前 proof 充当 last proof 计算下一个 proof，完全是可预测的，相当于考试前就知道试题，可以计算出后面所有的 proof，完全抢占记账权，轻轻松松发动 51%攻击毁掉这条区块链

所以，为了不能被预测到，可以在下一个 proof 的计算中，必须引入上一个区块独有的数据，比如 hash 值、时间戳等等，这样只有产生上一个区块后，相当于得到发号施令后才能开始猜测 proof，不会出现上面的考试前就已经知道试题的问题

我这里采用的策略是加入上一个区块的 hash 值，更改后的代码如下

```
def proof_of_work(self, last_block):
    """
    Simple Proof of Work Algorithm:
    - Find a number p' such that hash(pp') contains leading 4 zeroes
    - Where p is the previous proof, and p' is the new proof

    :param last_block: <dict> last Block
    :return: <int>
    """

    last_proof = last_block['proof']
    last_hash = self.hash(last_block)

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

    return proof

@staticmethod
def valid_proof(last_proof, proof, last_hash):
    """
    Validates the Proof

    :param last_proof: <int> Previous Proof
    :param proof: <int> Current Proof
    :param last_hash: <str> The hash of the Previous Block
    :return: <bool> True if correct, False if not.
    """

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

3.实验与现实的不同点

这次的实验用老师们经常用的一个词来形容就是叫“玩具”，向我们展示区块链的大致意思，然而实际过程比它复杂几万倍

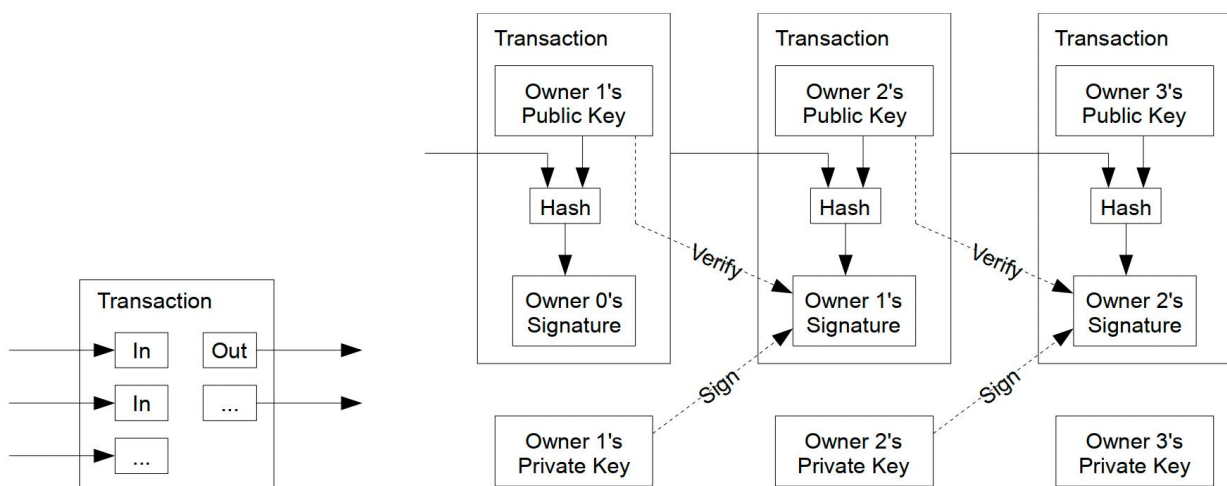
- 奖励不能无限发行。比特币的奖励是大概每 4 年减半，从最开始 25，减到 12.5，减到现在的 6.25，比特币的总数控制在 2100 万枚，如果奖励数一直是一样感觉是不合理的，设计到通货膨胀、货币贬值那些经济学规律

- 数据结构也不是很对，如同上面阐述的时间戳 timestamp 以及 proof of work，时间戳如果不送到 TSA 签名认证很容易 被冒充，难度也不能动态调整，一直是开头 0000 就行，如果所有节点的算力暴增或者暴降，无法动态地控制区块产生的时间间隔
- 这根本就不是 P2P 应用，那些交易和挖到的区块根本就没有进行广播
- 这个程序最大的问题应该无法进行交易，交易的过程非常复杂

比特币采用的是 **UTXO** 模型，并非账户模型，并不直接存在“余额”这个概念，余额需要通过遍历整个交易历史得来。获取余额需要扫描整个区块链，而当区块非常多的时候，这么做就会花费很长时间。并且，如果我们想要验证后续交易，也需要花费很长时间。而 **UTXO** 集就是为了解决这些问题，加快交易相关的操作。

交易的结构是非常复杂的，一笔比特币交易是一个包含输入值和输出值的数据结构，该数据结构植入了将一笔资金从起始点（输入值）转移至目标地址（输出值）的代码信息，输入值和输出值与账户或身份信息无关，里面涉及到非对称加密以及 **hash** 加密。

交易可以理解成一种被特定秘密信息锁定的一定数量的比特币，只有拥有者或知晓这个秘密信息的人可以解锁。



大小	字段	描述
4 字节	版本	明确这笔交易参照的规则
1-9 字节	输入数量	被包含的输入的数量
不定	输入	一个或多个交易输入
1-9 字节	输出数量	被包含的输出的数量
不定	输出	一个或多个交易输出
4 字节	时钟时间	一个 Unix 时间戳或者区块号

交易结构

还有一点也很有意思，矿工一般都会优先打包小费给的比较多的交易，因为那样的话他计算出一个区块获得的收益最高。区块记录的交易记录中，第一条交易通常是 **Coinbase** 交易，也就是矿工的挖矿奖励，后续交易都是用户的交易。

Block Reward	6.25000000 BTC
Fee Reward	0.04683806 BTC

Block Transactions ⓘ

Fee

0.00000000 BTC
(0.000 sat/B - 0.000 sat/WU - 206 bytes)
(0.000 sat/vByte - 179 virtual bytes)

6.29683806 BTC
1 Confirmations

Hash

d17d1d20a903516e8f748fc3acc591ea35bb6c9aa284189cd518c8...

2021-11-18 22:16

COINBASE (Newly Generated Coins)

➡

bc1qadv9a92in0i58hh6g0e9jeuz4y5cg0m0sj...

6.29683806 BTC

OP_RETURN

0.00000000 BTC

还有对交易信息的加密得到的 **Merkel** 根来进一步确保交易信息不被更改，椭圆曲线加密，全节点、挖矿节点、轻节点划分等等，现实中的过程比模拟实验要复杂太多，每一个小细节感觉都够我研究半天

4.对区块链的感想

这几天查了很多博客论坛资料，阅读了相关书籍，看了很多相关视频，读完了比特币白皮书，甚至去几个区块链网站查看实时数据，来了解区块链，花了非常多的时间，到头来还是一头雾水，我仔细分析了一下原因，主要还是因为区块链技术太过于庞大了，咱们上次实验的加密算法就仅仅是它的一小部分，还涉及到非常多的关于金融、算法、数学、数据结构、博弈论、计算机网络、密码学的知识，而且这方面的资料国内比较稀缺，国外网站的英文资料读起来还是有点费劲，我感受到了很大的挫败感，同时也感受到了中本聪的伟岸，如果时间足够的话，我非常想了解区块链、比特币的实现细节，但是由于别的课程学业压力大，可能就没那么多时间深究，非常遗憾。

但是依旧不得不感慨中本聪的智慧，比特币精妙绝伦的设计，将各类知识发挥到了极致，推动了比特币在全球范围内扩张，建立起一个新兴产业，很可能会颠覆人类几千年来支付信用体系，美国的不少 **APP**、网站都支持加密货币支付，也有不少国家将其认定为法定货币，我们国家在区块链上也有很多大动作。最近显卡价格的飙升居高不下，挖矿的

热潮，炒币的热潮，这就是比特币的魅力，无数人为之疯狂，大批人通过比特币一夜暴富。

但是也有不少人看衰区块链，这也是可以理解的，如果未来出现超大算力的量子计算机，到时候所有的加密算法恐怕都会失效，因为量子计算机算力太强了，直接在很短的时间内就能穷举出正确的原始数据。比特币的匿名性有时候也是它的缺点，不少公司被黑客攻击勒索，支付方式就是比特币，因为匿名性追踪起来比较困难，被一些不法分子利用了。

不过我开始留意到 **SHA256** 加密算法，它贯穿于密码学，通过它能创造出独一无二的数据指纹，非常神奇，它的单向性、雪崩性，我对他的内部加密过程也非常感兴趣。

网络与信息安全这门课，打开了我的求知欲和兴趣点，每次的实验我都特别感兴趣，也会情不自禁的花很多的时间去研究，通过这几次实验，确实学到了很多终生受益的知识。

5.实验的悲惨经历

还有一件特别悲惨的事情，就是我那天下午写完报告后，把文件给误删了，我报告的名字叫“**blockchain.docx**”，然后还有一个草稿叫“**bitcoin.docx**”里面存着查的资料，当时写完报告就想着把 **bitcoin** 那个文件给删了，可能是因为写报告写太久了人有点懵，结果删错了，把我报告直接给删了，也没留下备份，这都是因为我多年的强迫症，每次删东西都是直接删不过回收站的，我依稀还能感受到当时的绝望，一下午的心血说没就没了，这也算是一次经验教训吧，现在我写文档的时候都开着云端实时备份，再也不敢乱删文件了。好在脑子里还有印象，花了几个小时复盘出来了。

QQforMac官方体验群



起名快把我气死了
31条消息

救命 🙏

起名快把我气死了

macbook误删文件咋恢复啊

起名快把我气死了

写了一天的作业就这么没了

M1 Pro_12.0.1

没救了

FlintyLemming

没救了

FlintyLemming

养成配置 TimeMachine 的好习惯

冰冻大西瓜

误删也是到垃圾桶啊