Blockchain Lab

CHEN Hang

Exercise 1: Where is the pointer to the previous block stored?

The pointer to the previous block is stored in “Previous Hash”.

Exercise 2: Why is it important to sort the keys?

Because “sort keys=True” can sort the json file by key to make sure that each json file of the header blockchain has the attributes in order. If we don’t sort it, the json file will be sorted by the value, so we could not get the json file with the same format.

Exercise 3: Which of the header’s fields is unnecessary and redundant?

“index” is unnecessary and redundant because it just stores the number of block. But our blockchain is the chain type, we have already known its last block and next one, it means that it has already the order, so we don’t need to have a attribute to store it.

Exercise 5. Compare our header with the header of the Bitcoin. Which fields are missing? Describe briefly what they are used for.

The structure of Bitcoin is: Version, Previous Hash, Merkle Root, Timestamp, Bits, Nonce.

The missing fields is:

Version: The version of the block.

Merkle Root: All of the transactions in this block, hashed together. Basically provides a single-line summary of all the transactions in this block.

Bits: A shortened version of the Target.

Exercise 7: Which field is missing to make the transaction secured? Discuss a possible attack.

The field of signature of sender.

If this field is missing, the attacker can forge the sender's name to change the content of the transaction in order to transfer money to himself.

Exercise 11: The directory blocks, contains the serialization of some blocks. How many transactions are in each block?

Here we have the codes:

```
for i in range(0,10):
    file = "blocks/block" + str(i) + ".json"
    with open(file,'r') as load_f:
        block = read_block(json.load(load_f))
        print("The number of transactions in " + file + " is " +
str(len(block.transactions)))
with open("blocks/initial_block.json",'r') as load_f:
    block = read_block(json.load(load_f))
    print("The number of transactions in blocks/initial_block.json is " +
str(len(block.transactions)))
```

And the result is:

```
The number of transactions in blocks/block0.json is 46
The number of transactions in blocks/block1.json is 46
The number of transactions in blocks/block2.json is 9
The number of transactions in blocks/block3.json is 20
The number of transactions in blocks/block4.json is 26
The number of transactions in blocks/block5.json is 18
The number of transactions in blocks/block6.json is 19
The number of transactions in blocks/block7.json is 30
The number of transactions in blocks/block8.json is 28
The number of transactions in blocks/block9.json is 12
The number of transactions in blocks/initial_block.json is 6
```

Exercise 12: What is roughly the average size of a block for the Bitcoin?

From the data of the website "Quandl", we could compute that in recent 5 years, the average size of a block for the Bitcoin is about 0.75 MB, and in recent 50 days it is about 1 MB per block.

Exercise 13: Why is nonce the only parameter we want to be able to modify?

The "nonce" in a bitcoin block is a 32-bit (4-byte) field whose value is set so that the hash of the block will contain a run of leading zeros. For the proof of work, changing a little of data of the block will cause a large change of the hash value, so "nonce" is used to find the hash value and define the difficulty of work. What's more, The rest of the fields can not be changed, as they have a defined meaning in the block.

Exercise 15: What is the advantage of hashing only the header and not the entire block?

The header has contained all the information of this block because it has the field "Merkle Root" which presents the informations of the transactions, so it's enough to hash the header to ensure the security. So hashing only the header and not the entire block can reduce much memory of the informations of transaction and make the proof of work more easy.

Exercise 16: The structure of our header has a huge security problem. Can you guess it? Describe a possible attack.

Because in our defined header, there is no the field of "Merkle Root" , which hashes all of the transactions in this block. So if it exists, we can ensure that when we hash the header of the block, it has contained the information of the transactions and can't be changed(If we want to change the transactions, we need to change the other blocks). But if there is not "Merkle Root" of the header, the attacker can change the inforamations of transactions easily without modifying the header.

Exercise 18: For all blocks in the directory blocks to prove, prove it and give the nonce and the value of the hash function.

Codes:

```
for i in range(0,10):
    file = "blocks_to_prove/block" + str(i) + ".json"
    with open(file,'r') as load_f:
        block = read_block(json.load(load_f))
        print("For the file " + file + " :")
        block.make_proof_ready()
```

Results:

```
For the file blocks_to_prove/block0.json :
The nonce is: 1438
The hash value is:
0000aa66b0d1e8c3c7268f69d1ce2e39607c6ef33957a483c513b0372de1d2fe
For the file blocks_to_prove/block1.json :
The nonce is: 17773
The hash value is:
000078ebc04a8c827de21e71eadddb5ce59d647999a3638433403cf7241bed27
For the file blocks_to_prove/block2.json :
The nonce is: 70441
The hash value is:
0000234700122aca31d18901bb723fdc4cf7f71dd47ec55f11f71e3693dddb52
For the file blocks_to_prove/block3.json :
The nonce is: 135140
The hash value is:
0000180be1b98053bd2e0433c78d38d630afc5e45f91ede9dac86f4e704f1a80
For the file blocks_to_prove/block4.json :
The nonce is: 25067
The hash value is:
0000bef8730d5c78679f854d0771b3d7fd9fce926df2e9d52391d7331ab981b9
For the file blocks_to_prove/block5.json :
The nonce is: 37716
The hash value is:
000085656a163e42a67346007bab61ad890452f00832654c4020181d8afd140e
For the file blocks_to_prove/block6.json :
The nonce is: 22454
```

```
The hash value is:
0000e41bbe5a1cb64a40e2e9d59da8994abae6df9345ec115d77781484a58bd8
For the file blocks_to_prove/block7.json :
The nonce is: 27395
The hash value is:
0000a76aa2b0308782cedcf01a1958401bf2a219147588248dd8a2d0d3d74292
For the file blocks_to_prove/block8.json :
The nonce is: 188002
The hash value is:
00006e24a6ddccbb6423d7ddaae7161f79cf863d45337f2b4bc9c6a650fe83eb
For the file blocks_to_prove/block9.json :
The nonce is: 59304
The hash value is:
00009cc0b80cee2c556d93226e3802bb75c5e9c3ca7127d64165e833c605e8aa
```

Exercise 19: Take one block from the directory blocks to prove and observe what happens when you increase the number of requested starting zero in the proof. From which number of leading zeros does the computation of the proof takes more than one minute? How many leading zeros are required in the Bitcoin system?

For the "block0.json", when I increase the number of requested starting zero in the proof, the running time will also increase. The adding of the starting zero will increase the difficulty of the proof.

From 7 leading zeros, the computation of the proof takes more than one minute.

For the Bitcoin system, the number of the leading zeros is always increasing, from 2016 to 2018, the required number was from 17 to 19.

Exercise 23: For each blockchain in the directory blockchain wallets, compute the value of the wallet of each user. To do so, create a Blockchain object and call the add block method for each block.

Codes:

```
for i in range(0,10):
    block_chain = Blockchain()
    file = "blockchain_wallets/chain" + str(i) + ".json"
    with open(file,'r') as load_f:
        for block in read_chain(load_f.read()):
            block_chain.add_block(block)
        print("For the file " + file + " :")
        for (k,v) in block_chain.wallets.items():
            print(k + ": " + str(v))
```

Result:

```
For the file blockchain_wallets/chain0.json :
admin: 99999999999940
```

```
alice: 3.736930341079879
bob: 0.5199175472366541
fabian: 2.4929180882430533
nicoleta: 2.2420553507989878
jonathan: 40.50885230230437
julien: 10.499326370337057
For the file blockchain_wallets/chain1.json :
admin: 99999999999940
alice: 1.0499718441547072
bob: 21.24404655948255
fabian: 10.357225144475336
nicoleta: 19.16332750120396
jonathan: 1.9068199722888832
julien: 6.27860897839449
For the file blockchain_wallets/chain2.json :
admin: 99999999999940
alice: 4.83669096331737
bob: 6.235610499441814
fabian: 12.577800109443528
nicoleta: 0.7813231779182515
jonathan: 7.369820586914944
julien: 28.19875466296409
For the file blockchain_wallets/chain3.json :
admin: 99999999999940
alice: 2.6301404199753673
bob: 7.481801786732424
fabian: 3.114144934748701
nicoleta: 30.73825034412046
jonathan: 1.8162332886221757
julien: 14.219429225800877
For the file blockchain_wallets/chain4.json :
admin: 99999999999940
alice: 16.511951701327664
bob: 3.204551702235803
fabian: 17.459595055237774
nicoleta: 14.014437571117117
jonathan: 8.253641360454935
julien: 0.5558226096266657
For the file blockchain_wallets/chain5.json :
admin: 99999999999940
alice: 2.3484898074202034
bob: 11.805603895813888
fabian: 21.048340054844207
nicoleta: 0.7053221224742485
jonathan: 0.41059166819114157
julien: 23.681652451256323
For the file blockchain_wallets/chain6.json :
admin: 99999999999940
alice: 23.74148251549724
```

```
bob: 14.733496333101233
fabian: 5.235568257131697
nicoleta: 8.233503626015727
jonathan: 0.009932795761336584
julien: 8.046016472492784
For the file blockchain_wallets/chain7.json :
admin: 99999999999940
alice: 6.218826643518955
bob: 12.446261288514325
fabian: 19.54745363856009
nicoleta: 1.6442521356330002
jonathan: 0.8392269651807256
julien: 19.303979328592902
For the file blockchain_wallets/chain8.json :
admin: 99999999999940
alice: 1.2988516033107085
bob: 3.96102453987966
fabian: 11.70246704717524
nicoleta: 0.5485314025300652
jonathan: 4.252151186880825
julien: 38.23697422022352
For the file blockchain_wallets/chain9.json :
admin: 99999999999940
alice: 4.603391989607367
bob: 5.60605688842351
fabian: 4.856058056254167
nicoleta: 9.337103555411835
jonathan: 21.909844214499614
julien: 13.687545295803474
```

Exercise 25: For each blockchain in the directory blockchain incorrect, check if it is correct. If a blockchain is incorrect, give the index of the first incorrect block and if necessary the index of the first incorrect transaction.

Codes:

```
for i in range(0,10):
    file = "blockchain_incorrect/chain" + str(i) + ".json"
    with open(file,'r') as load_f:
        block_chain = Blockchain()
        index_error = 0
        print("For the file " + file + " :")
        for block in read_chain(load_f.read()):
            if not block_chain.add_block(block):
                print(" of the block " + str(index_error))
                break
            else:
                index_error += 1
```

Result:

```
For the file blockchain_incorrect/chain0.json :
The index of the first incorrect is in the transaction 12 of the block 1
For the file blockchain_incorrect/chain1.json :
The index of the first incorrect is in the transaction 23 of the block 1
For the file blockchain_incorrect/chain2.json :
The index of the first incorrect is in the transaction 31 of the block 1
For the file blockchain_incorrect/chain3.json :
The index of the first incorrect is in the transaction 13 of the block 1
For the file blockchain_incorrect/chain4.json :
The index of the first incorrect is in the transaction 10 of the block 1
For the file blockchain_incorrect/chain5.json :
The index of the first incorrect is in the transaction 18 of the block 2
For the file blockchain_incorrect/chain6.json :
The index of the first incorrect is in the transaction 25 of the block 1
For the file blockchain_incorrect/chain7.json :
The index of the first incorrect is in the transaction 35 of the block 2
For the file blockchain_incorrect/chain8.json :
The index of the first incorrect is in the transaction 13 of the block 2
For the file blockchain_incorrect/chain9.json :
The index of the first incorrect is in the transaction 27 of the block 2
```