UNIVERSITÀ DI PISA

MidTerm project

Morucci Davide mat. 548058

6/5/2022

Bitcoin Analysis

Describe how a real Bitcoin transaction is abstracted by a transaction in the dataset (which fields are eliminated, which are abstracted and how).

Compared to a real Bitcoin transaction, some changes have been made to the bitcoin dataset. In particular, in addition to the fact already mentioned that we have replaced the 256-bit HASH with numeric ids (both for input, output and transactions), the following changes are introduced:

1. Input

   Output number identifies the amount of output you want to spend, it has been removed

   ScriptSig Lenght indicating the length of the script, it has been removed

   ScriptSig (Signature Script) which contains information that must meet the spending conditions indicated in the PubKey Script, present in the output**.** The **ScriptSig is** the first part of the cryptographic signature to verify the authenticity of the input and is composed, in addition to the Signature Script, of the **public key**. In the dataset it was introduced in a different form as **scriptSigId** then simply as the ID of the corresponding signature.

   NSequence: Removed from input in the dataset

1. **Output**

   ScriptSig Lenght indicating the length of the script, has been removed

   ScriptSig (Signature Script) like what was done for the signature script in the inputs, in this case only the public key in the form of a numeric id has been kept, in order to simplify its management.

2. Transaction

   ProtocolV. not present within the transaction in the dataset

   Size. not present within the transaction in the dataset

LockTime. not present within the transaction in the dataset

The fields related to the **list of inputs and outputs** can be easily reconstructed within the dataset, exploiting the tx_id values present in both the Input and Output records.

Check if all the data contained in the dataset is consistent, and if some data is invalid, describe what is the problem of that data and remove it from the dataset.

In the dataset there are series of coinbase bitcoin transactions, i.e. transactions generated by the system as a reward for miners who have done useful work to allow the addition of a new block.

In the original version of the protocol, such "reward" transactions were worth 50BTC (then decreased over the following years).

In the dataset, however, there are coinbase transactions that have a reward greater than or less than 50 BTC, or at least a value different from it.

Considering that the last block analyzed within the dataset, appears to have been mined before the value of the reward was halved (25 BTC), these transactions were marked as invalid.

There are transactions in the dataset that have sums of input values that are less than sums of output values. This would mean generating bitcoins out of thin air, which is obviously not possible and is not allowed by the protocol.

These transactions were then located and removed from the dataset

Signatures related to the P2PKH mechanism were checked, to verify that the inputs were correctly correlated with the outputs used as a "spendable part".

In other words, it has been checked that the SIG of an input is equal to the PK of the output it tries to spend.

If there were transactions with different input and output values for SIG and PK, these were deemed invalid and removed.

It was also verified that there were no attempts at double spending by the nodes involved. In the case of double spending, the relative transaction has therefore been marked and deleted.

Finally, the outputs of the various transactions were verified, by checking for the possible presence of negative outputs (< 0). These outputs were thus marked, and their transactions deleted because they were invalid.

NB. Invalid transactions will produce outputs that are also invalid and therefore not to be considered; if these outputs are used in subsequent transactions, these transactions will also be considered invalid, thus creating chains of invalid transactions. These invalid transaction chains have therefore been removed.

Compute the total amount of UTXOs (Unspent Transaction Outputs) existing as of the last block of the data set, i.e. the sum of all the Transaction outputs balances on the UTXO set of the last block. Which UTXO (TxId, blockId, output index and address) has the highest associated value?

Calculating the total of UTXOs that have not been consumed and therefore still reside in the system at the end of the last block, is a total of:

Tx : 139532 Block: 90018 Output id : 169296 Address: 70054 Valore: 55000.0
Tot UTXO: 495501985646619 satoshi
Tot UTXO: 4955019.85646619 BTC

STATISTICS

The distribution of the block occupancy, i.e. of the number of transactions in each block in the entire period of time. Furthermore, show the evolution in time of the block size, by considering a time step of one month.

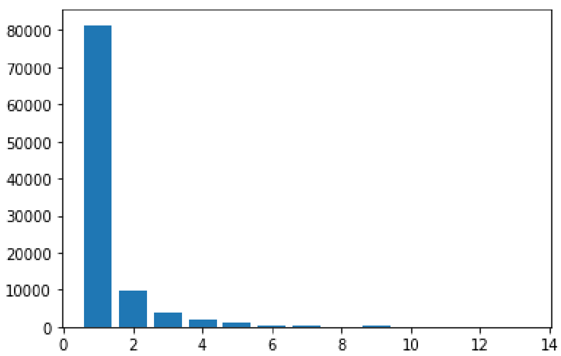

Figura 1

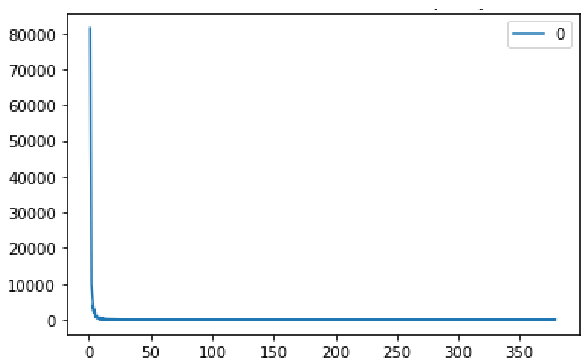

Figura 2

The distribution was calculated by considering along the vertical y-axis, the number of blocks that have a size equal to x (i.e. the size represented along the horizontal axis).

Two different graphs have been reported, the graph in figure 1 is a focus on the number of blocks having a size between 1 and 14, where we note the predominance of blocks with very small sizes; The graph on the right shows the distribution with respect to the total size.

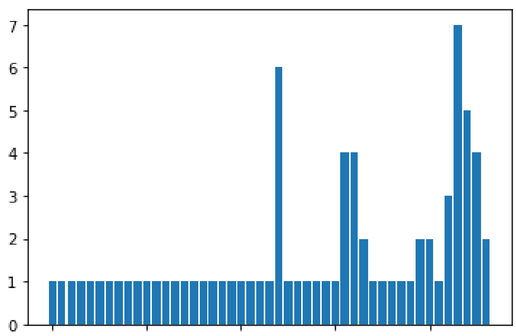

Figura 3

To consider the evolution in terms of time, of the size of the various blocks, the following metric was used:

averagely, if we consider that a block is mined every 10m or so, we count that in a month (1440 minutes a day x 30) about 4320 blocks are mined.

In this case, blocks a month apart were taken to represent the approximate distribution of block size within the dataset.

Obviously, the analysis can also be varied by taking specific reference periods, or precise months (always calculated in terms of mined blocks in a month).

The estimate shown in the graph in Figure 3 is an estimate made over the entire dataset

The total amount of bitcoin received from each public key that has received at least one COINBASE transaction, in the considered period, and show a distribution of the values

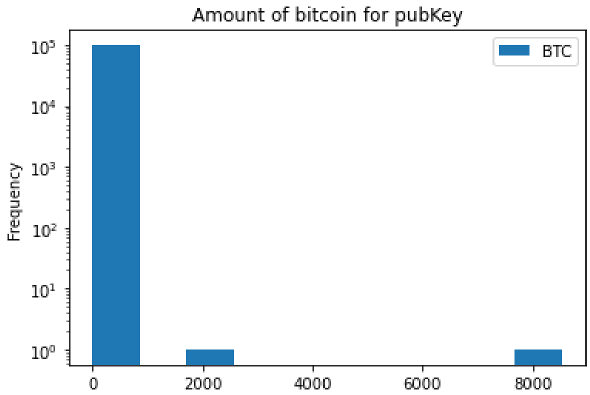

Figura 4

After calculating the total of bitcoins in possession of each public key, to represent the distribution of these values, the graph was drawn considering some simplifications, in particular the keys having an equal bitcoin value were considered as a single pubK.

In this case, the graph was then drawn where the number of (approximate) addresses having a bitcoin value equal to that specified along the x-axis is expressed along y is expressed.

The distribution of the fees spent in each transaction in the considered period.

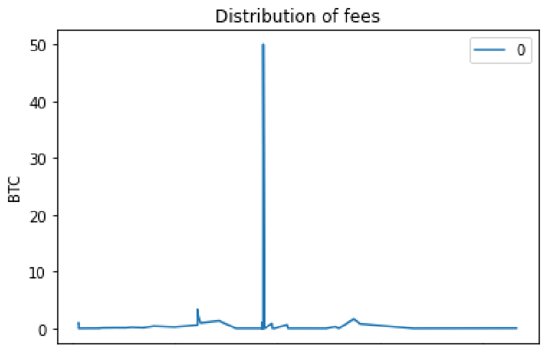

Figura 5

The graph in figure 5 shows a distribution of the various fees in each transaction distributed in the total of the period.

ADDITIONAL ANALYSIS

Taking advantage of the tools made available through the networkx library, it was possible to draw the graph of the addresses, or the graph through which it is possible to represent the flow of BTC between the various public keys, or addresses.

For performance reasons it was not possible to execute the plot of the graph in its entirety, since the number of nodes is very high.

However, it was still possible to calculate some useful metrics, such as the flow of bitcoins between two addresses for example, or alternatively the mass flow of BTC between two nodes.

In this case, two random nodes were chosen, No. 103919 and No. 155526 and the value of the max flow was calculated through the maximum_flow function, which gave a source node and a recipient node, Find a maximum single-commodity flow.

Flow_value = `402750000.0`

Other metrics taken as a reference concern the Degree and PageRank:

the Degree of a node basically indicates the number of arcs that this node possesses, in essence, in our case indicates with how many other nodes it shares an expense.

```
Degree of node 103919: 3
Degree of node 155526: 3
```

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links.

```
Degree of node 155526: 1.1699657033258848e-05
Degree of node 103919: 1.1786182517118949e-05
```

Kademlia

Assume a Kademlia network with ID size of 8 bits. The bucket size is k = 4.

The k-buckets of the peer with ID 11001010 are as follows:

k-Bucket 7: 01001111, 00110011, 01010101, 00000010

k-Bucket 6: 10110011, 10111000, 10001000

k-Bucket 5: 11101010, 11101110, 11100011, 11110000

k-Bucket 4: 11010011, 11010110

k-Bucket 3: 11000111

k-Bucket 2:

k-Bucket 1:

k-Bucket 0:

Messages from following nodes arrive in this given order: 01101001, 10111000, 11110001, 10101010, 11100011, 11111111 How do the buckets, the orderings in the buckets and the waiting lists change?

1) Message from 01101001:

   The node should be inserted in the bucket n. 7, but it is full.

   In this case the owner of the routing table pings the node which has been visited recently, which is node 01001111.

   Suppose that the node is not alive: in this case this note is removed from the bucket n.7, and the sender ID is added at the tail.

   k-Bucket 7: 00110011, 01010101, 00000010, 01101001

   k-Bucket 6: 10110011, 10111000, 10001000

   k-Bucket 5: 11101010, 11101110, 11100011, 11110000

   k-Bucket 4: 11010011, 11010110

   k-Bucket 3: 11000111

   k-Bucket 2:

   k-Bucket 1:

   k-Bucket 0:

2) Message from 10111000:

   The node already exists in bucket n. 6
   So, it is move to the tail of the list.

   k-Bucket 7: 00110011, 01010101, 00000010, 01101001

   k-Bucket 6: 10110011, 10001000, 10111000

   k-Bucket 5: 11101010, 11101110, 11100011, 11110000

   k-Bucket 4: 11010011, 11010110

   k-Bucket 3: 11000111

   k-Bucket 2:

   k-Bucket 1:

   k-Bucket 0:

3) Message from 11110001:

The id should be inserted in the bucket n. 5, but it is full.

The owner pings the last recently node visited which is 11101010.

Suppose it is alive, so the ID is moved to the tail of the list and the ID’s sender is discarded.

k-Bucket 7: 00110011, 01010101, 00000010, 01101001

k-Bucket 6: 10110011, 10001000, 10111000

k-Bucket 5: 11101110, 11100011, 11110000, 11101010

k-Bucket 4: 11010011, 11010110

k-Bucket 3: 11000111

k-Bucket 2:

k-Bucket 1:

k-Bucket 0:

4) Message from 10101010:

The node should be inserted in the bucket n.6.
The bucket is not full, and the ID is not already present in the bucket, so it will be added at the end of the list.

k-Bucket 7: 00110011, 01010101, 00000010, 01101001

k-Bucket 6: 10110011, 10001000, 10111000, 10101010

k-Bucket 5: 11101110, 11100011, 11110000, 11101010

k-Bucket 4: 11010011, 11010110

k-Bucket 3: 11000111

k-Bucket 2:

k-Bucket 1:

k-Bucket 0:

5) Message from 11100011:

The message is already present in bucket n.5, so it will be moved to the end of the list.

k-Bucket 7: 00110011, 01010101, 00000010, 01101001

k-Bucket 6: 10110011, 10001000, 10111000, 10101010

k-Bucket 5: 11101110, 11110000, 11101010, 11100011

k-Bucket 4: 11010011, 11010110

k-Bucket 3: 11000111

k-Bucket 2:

k-Bucket 1:

k-Bucket 0:

6) Message from 11111111:

The ID should be inserted in the bucket n. 5 but it is full.
A ping is sent to node 11101110.
Suppose it is alive, so it is moved to the end of the tail and the sender's ID is discarded

The message is already present in bucket n.5, so it will be moved to the end of the list.

k-Bucket 7: 00110011, 01010101, 00000010, 01101001

k-Bucket 6: 10110011, 10001000, 10111000, 10101010

k-Bucket 5: 11110000, 11101010, 11100011, 11101110

k-Bucket 4: 11010011, 11010110

k-Bucket 3: 11000111

k-Bucket 2:

k-Bucket 1:

k-Bucket 0:

Now the node detects that peer 11101110 cannot be reached anymore, what is the reaction?

In this case the node is removed from the list in the bucket.

Which addresses would the peer reply to a lookup looking for ID 11010010?

In the case we can assume the value a = 1.

According to the routing table, the peer will ask to node in the bucket n.4.

Nodes are:

➔ k-Bucket 4: 11010011, 11010110

According to the XOR metric we can compute:

11010011 XOR 11010010 = 00000001

11010110 XOR 11010010 = 00000100

So, the node selected will be the node 11010011 which has a XOR distance equal to 1

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import networkx as nx
import random
```

In [ ]:

```
df_inputs = pd.read_csv('inputs.csv')
df_outputs = pd.read_csv('outputs.csv')
df_transactions = pd.read_csv('transactions.csv')
```

In [ ]:

```
class Transazione:
    def __init__(self,id_trans,block_id,tot_input,tot_output):
        self.id_trans = id_trans
        self.block_id = block_id
        self.tot_input = tot_input
        self.tot_output = tot_output
        self.coinbase = False
        self.is_valid = True
        self.output = []
        self.input = []

    def printTx(self):
        print('id tx ' + str(self.id_trans),'block id ' + str(self.block_id),'tot input ' + str(self.tot_input),'tot output ' + st

    def delTx(self):
        self.id_trans = 0
        self.block_id = 0
        self.tot_input = 0
        self.tot_output = 0
        self.coinbase = False
        self.is_valid = True

class OutputTx:
    def __init__(self,id_output,is_valid):
        self.id_output = id_output
        self.is_valid = is_valid

class InputTx:
```

```python
    def __init__(self,id_input,is_valid):
        self.id_input = id_input
        self.is_valid = is_valid


class Input:
    def __init__(self,id_input,id_trans,sig_id,output_id):
        self.id_input = id_input
        self.id_trans = id_trans
        self.sig_id = sig_id
        self.output_id = output_id
        self.is_valid = True
        self.is_double = False

    def printInp(self):
        print('id input ' + str(self.id_trans),'id tx ' + str(self.id_trans),'sig_id ' + str(self.sig_id),'output id ' + str(self.

class Output:
    def __init__(self,id_output,id_trans,pk_id,value):
        self.id_output = id_output
        self.id_trans = id_trans
        self.pk_id = pk_id
        self.value = value
        self.is_valid = True
        self.is_utxo = True
        self.is_spent = 0

    def printOut(self):
        print('id output ' + str(self.id_output),'id tx ' + str(self.id_trans),'pk_id ' + str(self.pk_id),'value ' + str(self.valu
```

In [ ]:

```python
#transazioni
tx_array = []
for index_trans in df_transactions.index:
    id_transazione_attuale = df_transactions['id'][index_trans]
    block_id_transazione_attuale = df_transactions['block_id'][index_trans]
    tx_array.append(Transazione(id_transazione_attuale,block_id_transazione_attuale,0,0))

#output
output_array = []

for index_output in df_outputs.index:
    id_out = df_outputs['id'][index_output]
```

```python
    id_tx = df_outputs['tx_id'][index_output]
    pk_id = df_outputs['pk_id'][index_output]
    value = df_outputs['value'][index_output]
    output_array.append(Output(id_out,id_tx,pk_id,value))

input_array = []

#input
for index_input in df_inputs.index:
    id_in = df_inputs['id'][index_input]
    id_tx = df_inputs['tx_id'][index_input]
    sig_id = df_inputs['sig_id'][index_input]
    out_id = df_inputs['output_id'][index_input]
    input_array.append(Input(id_in,id_tx,sig_id,out_id))
```

In [ ]:

```python
#per ogni tx, aggiorno la lista degli input e output per quella tx
for index,out in enumerate(output_array):
    tx_array[out.id_trans-1].output.append(OutputTx(out.id_output,0))

for index,inp in enumerate(input_array):
    tx_array[inp.id_trans-1].input.append(InputTx(inp.id_input,0))
```

In [ ]:

```python
#transazioni coinbase valide e non valide

for index, inp in enumerate(input_array):
    if inp.sig_id == 0 and inp.output_id == -1:
        tx_array[inp.id_trans-1].coinbase = True

for index, out in enumerate(output_array):
    if tx_array[out.id_trans-1].coinbase:
           tx_array[out.id_trans-1].tot_output += out.value #ricompensa coinbase

k=0
for index,tx in enumerate(tx_array):
    if tx.coinbase == True:
        if  tx.tot_output != 5000000000 :
            tx.is_valid = False
            k+=1

print('Totale transazioni coinbase non valide: ' + str(k))
```

In [ ]:

```
#calcolo tot degli output
for index,out in enumerate(output_array):
    if tx_array[out.id_trans-1].coinbase == False:
        tx_array[out.id_trans-1].tot_output += out.value
```

In [ ]:

```
#controllo output negativi
for index,out in enumerate(output_array):
    if out.value < 0:
        out.is_valid = False
        tx_array[out.id_trans-1].is_valid = False
```

In [ ]:

```
#calcolo tot degli input
#output_array.sort(key=lambda x: x.id_output)
for index,inp in enumerate(input_array):
     if tx_array[inp.id_trans-1].coinbase == False:
        if inp.output_id == 265834: #anomalo
            continue
        tx_array[inp.id_trans-1].tot_input += output_array[inp.output_id-1].value
```

In [ ]:

```
#transazioni non valide per via di input < output
i=0
tx_array[15698].is_valid=False
for index,tx in enumerate(tx_array):
    if tx.coinbase == False:
        if tx.tot_input < tx.tot_output:
            tx.is_valid = False
            tx.printTx()
            i+=1
print('Tot transazioni non valide: ' + str(i))
```

In [ ]:

```
#controllo firme
i=0
for index,inp in enumerate(input_array):
    if inp.output_id == 265834: #anomalo
        continue
    if inp.output_id == -1 :
        continue
```

```python
    if inp.sig_id == -1 :
        continue
    if inp.sig_id == 0 :
        continue
    sign = output_array[inp.output_id-1].pk_id
    if sign == -1 :
        continue
    if sign != inp.sig_id:
        print(sign,inp.sig_id)
        i+=1
        tx_array[inp.id_trans-1].is_valid = False
print('Tot firme non valide: ' + str(i))
```

In [ ]:

```python
#output_array.sort(key=lambda x: x.id_output)
i=0
for index,inp in enumerate(input_array):
    if inp.output_id == 265834: #anomalo
        continue
    if inp.output_id == -1 :
        continue
    output_array[inp.output_id-1].is_spent +=1
    if output_array[inp.output_id-1].is_spent >1:
        inp.is_double = True
        i+=1
        inp.is_valid = False
        tx_array[inp.id_trans-1].is_valid = False
        inp.printInp()


print('Tot double spending: ' + str(i))
```

In [ ]:

```python
#invalid tx
invalid_tx_id = []
for index,tx in enumerate(tx_array):
    if tx.is_valid == False:
        invalid_tx_id.append(tx.id_trans)

print('Numero di tx non valide' + str(len(invalid_tx_id)))
```

In [ ]:

```python
for index,tx in enumerate(tx_array):
```

```
    if tx.is_valid == False:
        for index1,inp in enumerate(tx.input):
            input_array[inp.id_input-1].is_valid = False
        for index2,out in enumerate(tx.output):
            output_array[out.id_output-1].is_valid = False
```

In [ ]:

```
i=0
for index,inp in enumerate(input_array):
    if inp.output_id == 265834: #anomalo
        continue
    if inp.output_id == -1 :
        continue
    #controllo se riferisce output di tx non valida
    if tx_array[output_array[inp.output_id-1].id_trans-1].is_valid == False:
        output_array[inp.output_id-1].is_valid = False #setto anche output come non valido
        for index3,out1 in enumerate(tx_array[output_array[inp.output_id-1].id_trans-1].output):
            output_array[out1.id_output-1].is_valid = False
        inp.is_valid = False #setto lo stesso input come non valido
        tx_array[inp.id_trans-1].is_valid = False #la transazione che sfrutta input derivato da output non valido, sarà invalidata
        for index2,out in enumerate(tx_array[inp.id_trans-1].output):
            output_array[out.id_output-1].is_valid = False
            #print( output_array[out.id_output].id_output,output_array[out.id_output].is_valid)
        if inp.id_trans not in invalid_tx_id:
            invalid_tx_id.append(inp.id_trans)
            i+=1

print('Transazioni non valide con meccanismo a catena: ' + str(i))

print('Totale transazioni non valide: ' + str(len(invalid_tx_id)))
```

In [ ]:

```
#set all input and output for invalid tx as not valid
for index,tx in enumerate(tx_array):
    if tx.is_valid == False:
        for index1,inp in enumerate(tx.input):
            input_array[inp.id_input-1].is_valid = False
        for index2,out in enumerate(tx.output):
            output_array[out.id_output-1].is_valid = False
```

In [ ]:

```
input_dict = {}
for index,inp in enumerate(input_array):
```

```python
    if inp.is_valid:
        input_dict[inp.id_input] = (inp.id_trans,inp.sig_id,inp.output_id)

output_dict = {}
for index,out in enumerate(output_array):
    if out.is_valid:
        output_dict[out.id_output] = (out.id_trans,out.pk_id,out.value,0)


tx_dict = {}
for index,tx in enumerate(tx_array):
    if tx.is_valid:
        tx_dict[tx.id_trans] = (tx.block_id,0) #id,tot_utxo
```

In [ ]:

```python
#calcolo totale del valore di UTXO nel sistema
for key in input_dict:
    if input_dict[key][2] != -1:
        if input_dict[key][2] == 265834:
            continue
        y = list(output_dict[input_dict[key][2]])
        y[3] += 1
        output_dict[input_dict[key][2]] = y #1 se non e utxo, 0 altrimenti

tot_utxo = 0
top_utxo = (0,0) #trans,value
for key in output_dict:
    if output_dict[key][3] == 0:
        y = list(tx_dict[output_dict[key][0]])
        if output_dict[key][2] > 0 :
            y[1] += output_dict[key][2]
        elif output_dict[key][2] < 0 :
            print(output_dict[key][2],'chiave output: ' + str(key))
            y[1] += 0
        tx_dict[output_dict[key][0]] = y

for key in tx_dict:
    if tx_dict[key][1] > top_utxo[1]:
        top_utxo = (key,tx_dict[key][1]) #id tx, valore utxo
    if tx_dict[key][1] > 0:
        tot_utxo += tx_dict[key][1]

out_id = 0
```

```
out_pk_id = 0
for key in output_dict:
    if top_utxo[0] == output_dict[key][0]:
        out_id = key
        out_pk_id = output_dict[key][1]

print('Tx : ' + str(top_utxo[0]),'Blocco: ' + str(tx_dict[top_utxo[0]][0]),'Output id : ' + str(out_id),
      'Address: ' + str(out_pk_id),'Valore: ' + str(top_utxo[1]* 10**(-8)))
print('Tot UTXO: ' + str(tot_utxo) + ' satoshi')
tot_utxo = tot_utxo * 10**(-8)
print('Tot UTXO: ' + str(tot_utxo) + ' BTC')
```

In [ ]:

```
#the distribution of the block occupancy, i.e. of the number of transactions in each block in the entire period of time.
block_occupancy = {} #key = n blocco, dim
for key in tx_dict:
    block = tx_dict[key][0]
    if block in block_occupancy:
         block_occupancy[block] += 1
    else:
        block_occupancy[block] = 1

h=[] #index blocco
k=[] #dim blocco

for key in block_occupancy:
        h.append(key)
        k.append(block_occupancy[key])

block_occupancy_dim = {} #key = dim
for key in block_occupancy:
    chiave = block_occupancy[key]
    if chiave in block_occupancy_dim:
         block_occupancy_dim[chiave] += 1
    else:
        block_occupancy_dim[chiave] = 1

x=[] #dim
y=[] #num
for key in block_occupancy_dim:
        x.append(key)
        y.append(block_occupancy_dim[key])
```

```
#support dicts
occ1 = {}
for key in block_occupancy_dim:
    if block_occupancy_dim[key] not in occ1:
        occ1[block_occupancy_dim[key]] = key

occ2 = {}
for key in occ1:
    occ2[occ1[key]] = key

df = pd.DataFrame.from_dict(occ2,orient='index')
df.plot(title=" Distribution of the block occupancy",  ylabel="Number of Blocks",xlabel="Transactions",kind="line")
```

In [ ]:

```
#show the evolution in time of the block size, by considering a time step of one month.
#considerando un tempo di 30g, e considerando che in media un blocco viene aggiunto alla bc ogni 10 minuti
#in media, il numero di blocchi minati in un mese di tempo è 4320
block_occupancy_month_tot = {}
i=0
for key in block_occupancy:
    if i == 0:
        block_occupancy_month_tot[key] = block_occupancy[key]
    i+=1
    if i == 4320:
        i=0
        block_occupancy_month_tot[key] = block_occupancy[key]

x=[] #dim
y=[] #num
i=0
for key in block_occupancy_month_tot:
        x.append(i)
        i+=1
        y.append(block_occupancy_month_tot[key])

plt.bar(x,y)
plt.show()

df = pd.DataFrame.from_dict(block_occupancy_month_tot,orient='index')

df.plot(title="Distribution of fees",
    ylabel="BTC",
```

```python
    kind = 'hist'
    )
```

In [ ]:

```python
#the total amount of bitcoin received from each public key that has
#received at least one COINBASE transaction, in the considered period, and show a distribution of the value
coinbase_tx = {}

for key in input_dict:
    if input_dict[key][2] == -1 and input_dict[key][1] == 0:
        coinbase_tx[input_dict[key][0]] = 0 #aggiungo il numero della tx

account_ = {}

for key in output_dict:
    if output_dict[key][0] in coinbase_tx:
        if output_dict[key][1] not in account_:
            account_[output_dict[key][1]] = 0 #se non c'è già l'account, lo aggiungo

tot_for_account = {}
for key in output_dict:
    if output_dict[key][1] in account_:
        if output_dict[key][1] not in tot_for_account:
            tot_for_account[output_dict[key][1]] = output_dict[key][2]
        else:
            tot_for_account[output_dict[key][1]] += output_dict[key][2]


for key in tot_for_account:
    tot_for_account[key] = tot_for_account[key]*10**(-8)


top = (0,0)
for key in tot_for_account:
    if tot_for_account[key] > top[1]:
        top = (key,tot_for_account[key])

tot = {}
for key in tot_for_account:
    if tot_for_account[key] not in tot:
        tot[tot_for_account[key]] = key
print(len(tot))
```

```
df = pd.DataFrame.from_dict(tot_for_account,orient='index',columns=['BTC'])
df.plot(title="Amount of bitcoin for pubKey",ylabel='BTC',kind='hist', logy=True)
```

In [ ]:

```
#the distribution of the fees spent in each transaction in the considered period.
coinbase_tx = {}

for key in input_dict:
   # print(input_dict[key])
    if input_dict[key][2] == -1 and input_dict[key][1] == 0:
        coinbase_tx[input_dict[key][0]] = 0 #aggiungo il numero della tx

tx = {}
for key in tx_dict:
    if key not in coinbase_tx:
        tx[key] = (0,0,0) #input,output,fee

for key in output_dict:
    if output_dict[key][0] in tx:
        y = list(tx[output_dict[key][0]])
        y[1] +=  output_dict[key][2]
        tx[output_dict[key][0]] = y

for key in input_dict:
    if input_dict[key][0] in tx:
        y = list(tx[input_dict[key][0]])
        y[0] +=  output_dict[input_dict[key][2]][2]
        tx[input_dict[key][0]] = y

#compute the fees
for key in tx:
    k = list(tx[key])
    k[2] =  (k[0] - k[1])*10**(-8)
    tx[key] = k

#support dicts
tx2 = {}
for key in tx:
    tx2[key] = tx[key][2]

tx3 = {}
for key in tx2:
```

```python
    if tx2[key] != 0:
        tx3[key] = tx2[key]

df = pd.DataFrame.from_dict(tx3,orient='index')

df.plot(title="Distribution of fees",
    ylabel="BTC",
    )
```

In [ ]:

```python
input_dict2 = {}
for index,inp in enumerate(input_array):
    if inp.is_valid:
        input_dict2[inp.id_input] = (inp.id_trans,inp.sig_id,inp.output_id)

output_dict2 = {}
for index,out in enumerate(output_array):
    if out.is_valid:
        output_dict2[out.id_output] = (out.id_trans,out.pk_id,out.value,0)

tx_dict2 = {}
for index,tx in enumerate(tx_array):
    if tx.is_valid:
        tx_dict2[tx.id_trans] = (tx.block_id,0,[],[]) #id,tot_input,input,output

for key in input_dict2:
    tx_dict2[input_dict2[key][0]][2].append(key)

for key in output_dict2:
    tx_dict2[output_dict2[key][0]][3].append(key)
```

In [ ]:

```python
for key in input_dict2:
    y = list(tx_dict2[input_dict2[key][0]])
    if input_dict2[key][2] == -1:
        continue
    y[1] += output_dict2[input_dict2[key][2]][2]
    tx_dict2[input_dict2[key][0]] = y
```

In [ ]:

```python
G = nx.Graph()
```

```
for key in input_dict2:
    if input_dict2[key][2] == -1:
        valore = 5000000000
    else:
        valore = output_dict2[input_dict2[key][2]][2]
    y = tx_dict2[input_dict2[key][0]][3] #prendo array output
    for elem in y:
        if tx_dict2[output_dict2[elem][0]][1] == 0:
            formulina = output_dict2[elem][2]*(valore/5000000000)
        else:
            formulina = output_dict2[elem][2]*(valore/tx_dict2[output_dict2[elem][0]][1])
        G.add_edges_from([(str(input_dict2[key][1]), str(output_dict2[elem][1]), {"weight" : formulina})])
```

In [ ]:

```
list_nodes = list(G.nodes)
a = random.choice(list_nodes)
b = random.choice(list_nodes)
#node choose for the test were 103919 and 155526
print(a,b)
flow_value, flow_dict = nx.maximum_flow(G,a,b,capacity='weight')

print(flow_value)
```

In [ ]:

```
print(a,b)
flow_value = nx.maximum_flow_value(G,a,b,capacity='weight')
```

In [ ]:

```
#One basic metric for a node is its degree: how many edges it has.
print('Degree of node 103919: ' + str(G.degree["103919"]))
print('Degree of node 155526: ' + str(G.degree["155526"]))

pageranks = nx.pagerank(G) # A dictionary
print('Degree of node 155526: ' + str(pageranks["155526"]))
print('Degree of node 103919: ' + str(pageranks["103919"]))
```