



**UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA**

MACROAREA DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

CORSO DI LAUREA IN INFORMATICA

A.A. 2020/2021

Tesi di Laurea

La Blockchain di Bitcoin: strumenti e tecniche per
l'estrazione e l'analisi dei dati

RELATORE

Prof. Francesco Pasquale

CANDIDATA

Giulia Pascale

Indice

Introduzione	1
1 Introduzione al protocollo Bitcoin	4
1.1 Perché è nato Bitcoin e il movimento Cypherpunk	4
1.1.1 Il movimento Cypherpunk	4
1.2 Bitcoin e bitcoin	5
1.2.1 Caratteristiche di bitcoin:	5
1.3 Come funziona	7
1.3.1 Hashing	7
1.3.2 Signature, Public Key e Secret Key, Wallet bitcoin	8
1.4 La sua nascita e il creatore: Satoshi Nakamoto	9
1.5 Blockchain	10
1.5.1 Cos'è la Blockchain?	10
1.5.2 Blocchi	11
1.6 Full Node	13
1.7 Mining e la Proof of Work	14
1.7.1 Il processo del mining	14
1.7.2 Creare un blocco e la Proof of Work	16
1.7.3 Mining Farm	17

1.7.4	Problema dei generali bizantini	17
1.7.5	51% attack	19
1.8	Breve introduzione alle Transazioni	19
1.9	Modello UTXO	20
1.9.1	Struttura di una Transazione	21
1.10	Fee delle Transazioni	22
2	Strumenti per l'estrazione dei dati nella Blockchain di Bitcoin	23
2.1	Strumento utilizzato: Rusty-blockparser	23
2.1.1	File csv	24
2.1.2	Gestione e generazione file	25
2.1.3	Difficoltà e limiti riscontrati	27
2.2	Analisi file csv con codice Python	28
2.3	Macchina utilizzata	29
3	Analisi dati sulla Blockchain di Bitcoin	30
3.1	Periodi di interesse	30
3.1.1	Divisione temporale dei blocchi	31
3.2	Analisi dei balance degli utenti	32
3.2.1	Balance complessivi	33
3.3	Analisi del valore delle UTXO	34
3.4	Analisi balance e UTXO	35
3.5	Analisi Blocksize	36
3.5.1	Blocksize	36
3.5.2	Risultati analisi	37
3.6	Analisi Transazioni nei blocchi	38

3.7	Analisi UTXO: per quanto tempo una transazione resta UTXO? . . .	39
3.7.1	Ricerca Binaria	40
3.7.2	Nella Blockchain Bitcoin	41
3.8	Analisi Coinbase	42
3.9	Analisi Miner	44
4	Conclusioni e sviluppi futuri	46
	Elenco delle figure	49

Introduzione

Da diversi anni le criptovalute hanno iniziato ad affermarsi sempre di più, prima tra tutte il **bitcoin**. Una delle caratteristiche principali di questa criptovaluta sta nel fatto che non esiste un registro dati come siamo abituati a pensarlo. Il sistema attuale si fonda sulle Banche, che hanno enormi server contenenti i dati di tutte le persone e al quale può accedere solo il personale bancario addetto.

Nel caso del bitcoin il sistema funziona diversamente: tutti i dati relativi alle transazioni sono di dominio pubblico e sono contenuti nella Blockchain che è un registro contenuto in ogni *full-node* della rete. Accedere ai *file* con i dati è semplice, ma non lo è altrettanto leggerne il contenuto. I *file* non sono criptati ma non esistono programmi per la loro analisi. Un ottimo strumento per l'utente interessato può essere il sito *Blockchain.com* che, nella sezione *explorer*, dà accesso a chiunque a molti dati sulle principali criptovalute come bitcoin (BTC), Ethereum (ETH) e Bitcoin Cash (BCH). Se si vuole approfondire lo studio dei *file* della Blockchain, il primo passo sarà scaricare l'intera Blockchain di Bitcoin nella propria macchina che deve avere a disposizione almeno 400 GB di memoria. Una volta ottenuti i dati occorre lo strumento giusto per leggerli. In questo lavoro è stato utilizzato un programma chiamato **Rusty-blockparser**, disponibile su GitHub [9] a chiunque volesse cimentarsi nell'analisi di Bitcoin.

Una volta approfondita la Blockchain di Bitcoin sono scaturite diverse domande a cui

questo lavoro si pone l'obiettivo di rispondere o di dare un'idea da cui partire per futuri sviluppi. In particolare è stato possibile approfondire la quantità di bitcoin posseduta in media dagli utenti e le transazioni non spese che compongono l'*input*. Di queste ultime è stata analizzata la variazione del loro ammontare e il loro tempo di risposta. Negli anni la popolarità del bitcoin è cresciuta, e con essa il numero di transazioni effettuate. Di conseguenza è stata investigata anche la variazione della *size* di ogni blocco, dato che ognuno di essi contiene informazioni riguardanti la transazione che conia nuova moneta, detta Coinbase, e le transazioni con cui gli utenti si scambiano moneta.

Inoltre è stata approfondita la figura dei *miner*, ovvero coloro che si occupano di validare le transazioni effettuate dagli utenti. In cambio del loro lavoro essi ricevono una *reward* generata ad ogni blocco.

Nonostante i discreti mezzi a disposizione, per motivi di complessità computazionale alcune analisi non sono state realizzate sull'intera Blockchain di Bitcoin, ma sono state fatte a campione, mentre per altre sono stati proposti dei prototipi.

Organizzazione della Tesi

Il Capitolo 1 fornisce un veloce approfondimento sull'argomento Blockchain di Bitcoin, soffermandosi su **cos'è** e **perché** esiste nella società moderna, per poter contestualizzare le analisi fatte successivamente.

Rusty-blockparser e analisi dati

Il capitolo 2 entra nello specifico degli strumenti utilizzati per derivare *file* dalla Blockchain di Bitcoin, che siano leggibili dai programmi più in uso, e nei codici scritti in Python per analizzarli.

Infine il Capitolo 4 sulle conclusioni evidenzia alcune limitazioni computazionali riscontrate e le problematiche relative all'analisi di alcuni dati.

Capitolo 1

Introduzione al protocollo Bitcoin

In questo primo capitolo verrà introdotta brevemente la storia che ha portato alla nascita e allo sviluppo del bitcoin. A seguire verranno illustrati i preliminari tecnici su cui è basato il protocollo Bitcoin.

1.1 Perché è nato Bitcoin e il movimento Cypherpunk

Credo sia necessario, per capirne e approfondire alcuni aspetti, capire prima perché è nato Bitcoin. Da dove è nata l'esigenza di definire un protocollo che ci permettesse di avere una moneta così diversa da come l'avevamo sempre immaginata?

1.1.1 Il movimento Cypherpunk

Nel Manifesto del movimento Cypherpunk, datato 9 marzo 1993 e firmato da Eric Hughes, si esprime la preoccupazione di proteggere la propria **privacy** in un momento in cui si stava entrando nell'era elettronica. Se siamo quello che compriamo, chi ha accesso alle informazioni sui nostri conti bancari e/o su cosa compriamo sa esattamente chi siamo, cosa vogliamo sentirci dire e come manipolarci. Al tempo era un'intuizione di quello che poi sarebbe successo 30 anni dopo alla nostra società, in

cui la privacy forse è utopia.

Nel manifesto Hughes si preoccupa di mettere in chiaro la differenza tra privacy e segretezza. Quando si cerca privacy non è perché si ha qualcosa da nascondere, ma perché "nella maggior parte dei casi l'identità personale non è rilevante". Se una persona è interessata ad acquistare una rivista, non importa chi è, importa che sia in possesso di abbastanza soldi per potersela permettere. A soluzione del problema viene proposta la crittografia, per rimuovere informazioni personali dalla sfera pubblica.

Non va confuso con il Cyberpunk, un movimento che si concentra più sulla tecnologia e sul fatto che chi la possiede ha più potere.

"Privacy is necessary for an open society in the electronic age. Privacy is not secrecy. A private matter is something one doesn't want the whole world to know, but a secret matter is something one doesn't want anybody to know. Privacy is the power to selectively reveal oneself to the world." Da: A Cypherpunk's Manifesto [11]

1.2 Bitcoin e bitcoin

Con bitcoin, scritto con la lettera minuscola, si intende la moneta, o meglio la crypto-valuta, pseudoanonima e decentralizzata. Non esiste una moneta fisica, esistono solo i dati associati alle transazioni di essa.

Bitcoin, scritto con la lettera maiuscola, è il protocollo, l'insieme di regole che definiscono le modalità di interazione.

1.2.1 Caratteristiche di bitcoin:

- **Decentralizzato:** non esiste un Ente centrale che emette la valuta e la gestisce.

Le operazioni di scambio della valuta sono effettuate da più nodi e i dati sono

replicati su ogni nodo che ha il protocollo Bitcoin al suo interno. Gli utenti si scambiano la valuta senza un Ente centrale intermediario, tramite un network peer-to-peer.¹

- **Economia deflazionaria:** bitcoin ha 2 regole per quanto riguarda la valuta:
 - il numero massimo di monete emesso sarà 21 milioni.
 - ogni 4 anni la *reward* che i minatori ricevono per il loro lavoro di convalidazione dei blocchi verrà dimezzata. [1.7]

Inoltre bisogna ricordare che il valore continuerà ad aumentare fintanto che gli utenti saranno interessati a far parte del network e ad usare bitcoin.

- **Distribuito:** non esiste una macchina centrale specializzata per l'archiviazione dei dati. La rete Bitcoin è formata da migliaia di nodi, tutti allo stesso livello, che possiedono gli stessi diritti e una replica degli stessi dati.
- **Dati sensibili:** bitcoin è una moneta pseudoanonima. Scelta una transazione a caso di bitcoin, non si sa chi sono i due interlocutori della transazione, e non importa saperlo. Quello che si sa è da quale *address* è partita la transazione e quali sono i destinatari. Non è possibile ricollegare gli *address* a delle persone fisiche.
- **Politiche monetarie:** bitcoin si pone l'obiettivo di non essere sotto il controllo esercitato dalle banche centrali. Non a caso è nato nel 2009, dopo la crisi finanziaria del 2008.

¹Una rete peer-to-peer, o abbreviata P2P, è una rete in cui non esiste una gerarchia nei nodi, ognuno di essi è equivalente e può funzionare sia come server che client.

1.3 Come funziona

1.3.1 Hashing

La funzione Hash Crittografica mappa stringhe di lunghezza arbitrarie in stringhe di lunghezza fissa. La funzione è $H : \{0, 1\}^* \rightarrow \{0, 1\}^b$ con $b \in \mathbb{N}$ e nel caso di Bitcoin viene utilizzata la SHA256, quindi $b = 256$.

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^{256}$$

Quindi è possibile interpretare una stringa di 256 bit in un numero appartenente all'intervallo $[0, 2^{256}]$. È una funzione deterministica: a dato input, corrisponde sempre lo stesso output. Questo è un problema "facile" computazionalmente parlando, dato che si parla di un problema lineare nella lunghezza di x , quindi $H(x)$ si calcola velocemente. Una funzione Hash crittografica deve soddisfare le seguenti proprietà:

- **Preimage resistance:** La funzione inversa non può essere calcolata in nessun modo banale. Infatti per la *preimage resistance*, dato un qualsiasi output y non deve essere possibile risalire all'input x tale che $H(x) = y$, se non con il metodo Brute Force. Data la lunghezza di y , nel caso di Bitcoin anche l'approccio Brute Force risulta impossibile da attuare in tempi utili.
- **Second preimage resistance:** Dato $x \in \{0, 1\}^*$ tale che $H(x) = y$, non è possibile calcolare in modo banale una $\hat{x} \in \{0, 1\}^*$ tale che $x \neq \hat{x}$ e $H(\hat{x}) = y$.
- **Collision Resistance:** Deve essere difficile trovare due input che hanno lo stesso output. Trovare $x_1, x_2 \in \{0, 1\}^*$ e $x_1 \neq x_2$ tale che $H(x_1) = H(x_2)$.

Ad oggi non si hanno funzioni che hanno queste proprietà con certezza, si può solo dire di avere delle candidate, come la SHA-256 utilizzata nel protocollo Bitcoin. Molte funzioni sono state "rotte" nel corso degli anni, cioè è stato dimostrato che non garantivano una delle proprietà sopra citate, come ad esempio la SHA-1 [17].

1.3.2 Signature, Public Key e Secret Key, Wallet bitcoin

Signature

Nel protocollo Bitcoin, una **signature** può essere vista come una firma che l'utente lascia su un insieme di dati per provare la sua identità. Per poterla utilizzare sono necessarie tre funzioni: *generateKey* (che si occupa di generare la chiave privata e la chiave pubblica), *sign* (che prende in input la chiave privata e il messaggio da firmare, restituendo un output diverso) e la funzione *verify* (che verifica la veridicità della firma).

Public Key e Secret Key

Nel protocollo Bitcoin si utilizza un sistema di crittografia asimmetrico, perché utilizza una coppia di chiavi chiamate **secret key** e **public key** (chiave privata e chiave pubblica). La secret key è una stringa che serve a firmare e trasferire bitcoin al nuovo proprietario. La public key viene generata dalla secret key tramite un processo che non è reversibile. L'unico modo per risalire alla secret key sarebbe tramite un approccio Brute Force, di conseguenza non è attuabile in tempi utili. Questa coppia di chiavi serve all'utente per poter scambiare moneta all'interno dell'ecosistema Bitcoin: la secret key serve a firmare la transazione che si vuole fare, che sarà verificata tramite la public key dalla rete Bitcoin.

Wallet bitcoin

Un **wallet** bitcoin è paragonabile ad un "portachiavi", infatti è un insieme di coppie di chiavi: la chiave pubblica e la chiave privata, che ci permette di gestire i bitcoin interagendo con la Blockchain. I wallet possono essere definiti **hot** o **cold**, in base a come operano. Un hot wallet è connesso ad internet, cosa che lo rende facile e più veloce da utilizzare, ma vulnerabile ad attacchi informatici, i quali causano perdita di dati. Un cold wallet non è gestito da un dispositivo connesso ad internet, cosa che lo rende più sicuro.[8]

1.4 La sua nascita e il creatore: Satoshi Nakamoto

Il primo articolo che presentò Bitcoin intitolato "Bitcoin: A Peer-to-Peer Electronic Cash System" [14], fu pubblicato il 31 Ottobre del 2008 su bitcoin.org, un dominio registrato nell'Agosto dello stesso anno. Questo articolo è ancora oggi consigliato per i primi approcci al protocollo Bitcoin e negli anni è stato tradotto in innumerevoli lingue, a testimonianza di quanto bitcoin sia diventato un fenomeno mondiale. [3] Questo articolo (che, volendo essere precisi, è un *whitepaper*, cioè un articolo senza revisione di terzi) riporta la firma di Satoshi Nakamoto, ma non sappiamo se sia una singola persona o se rappresenta un gruppo di persone. In ogni caso è lui che ringraziamo per il suo lavoro nella nascita e crescita di Bitcoin.

La prima transazione nella storia di Bitcoin è avvenuta il 3 Gennaio 2009, ma solo sei giorni dopo, il 9 Gennaio 2009 è stata rilasciata una prima versione del codice *open-source*. Non si hanno più notizie di Nakamoto dal Dicembre 2010, quando scrisse per l'ultima volta sul sito *bitcointalk.org*, ancora oggi molto usato per lo scambio di consigli nel mondo dei sviluppatori o solo curiosi di Bitcoin. Sul sito è scoprire leggere

che l'*account* non è più stato attivo da allora. [5]

1.5 Blockchain

Nel caso dei bitcoin, il Public Ledger chiamato "Blockchain", è decentralizzato. Questo significa che il registro pubblico che contiene i dati di tutte le transazioni avvenute fin dalla nascita della criptovaluta, è in possesso di ogni utente che ha Bitcoin Core [4] installato nel proprio dispositivo. Al contrario, le moderne istituzioni finanziarie sono sistemi centralizzati, ovvero i dati sono in un unico database controllato e gestito dall'istituzione stessa.

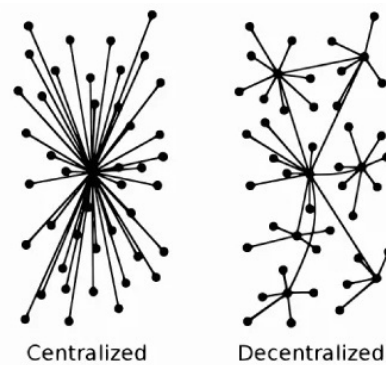


Figura 1.1: Sistema Centralizzato e Decentralizzato

La Blockchain è pubblica e chiunque può accedere alla lista di transazioni effettuate e al balance di un indirizzo specifico, è impossibile però ricollegare questo indirizzo ad una identità reale, proprio in virtù della volontà di tenere le proprie informazioni private, ma non segrete (come spiegato in 1.3.1).

1.5.1 Cos'è la Blockchain?

La Blockchain può essere letteralmente vista come una "catena di blocchi", infatti è formata da una serie di Blocchi, impilati uno sull'altro, al cui interno sono salvate

le transazioni avvenute. Capita spesso di vedere graficamente i blocchi allineati in orizzontale (come in Fig.1.2) per rendere di facile lettura le immagini, ma è più corretto immaginarli impilati in verticale, come una torre. Infatti all'interno di ogni blocco c'è il parametro "altezza". Una transazione che è all'interno di un blocco è stata validata, significa che è andata "a buon fine". Ogni blocco non solo conferma le transazioni al suo interno, ma anche le transazioni all'interno dei blocchi validati in precedenza. Questo significa che ogni blocco rappresenta una ulteriore conferma per tutti i blocchi precedenti. È convenzione pensare che dopo 6 conferme si può considerare una transazione immutabile. La verità è che una transazione non è mai immutabile al 100%, ma anche che è difficile che una transazione inserita in un blocco venga modificata prima che ci sia una sola conferma. Il numero di conferme da aspettare dipende da quanto l'utente è disposto a rischiare. Più bitcoin sono coinvolti nella transazione, più conferme aspetterà prima di considerarla andata a buon fine.

1.5.2 Blocchi

Come già detto, ogni blocco ha al suo interno una lista di transazioni che sono state validate. Il concetto di transazione e i dati all'interno di ogni transazione saranno approfonditi più avanti (1.8). In questo paragrafo si approfondiranno i parametri all'interno di ogni blocco. Un blocco viene minato all'incirca ogni 10 minuti e al suo interno è presente il campo "*previous block hash*", che fa riferimento all'hash del blocco precedente ad esso. (Fig.1.2)

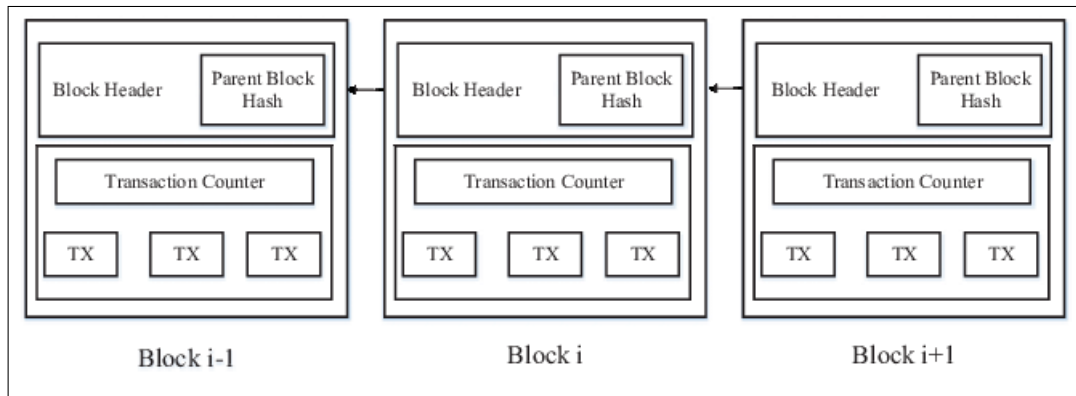


Figura 1.2: Esempio logico di blocchi consecutivi all'interno della blockchain

Alcuni dei suoi parametri sono il numero di conferme, la sua *size*, il *timestamp* che ci permette di sapere il momento esatto in cui è stato minato, la *versione*, la *merkle root*, il *nonce*, il *miner* (se si conosce) e il Block Reward. Tra gli altri parametri all'interno del blocco c'è la sua altezza all'interno della blockchain, che equivale al numero di nodi minati fino a quel momento. Attualmente la blockchain ha superato l'altezza di 700.000 blocchi. Il numero di conferme che ha un blocco, equivale al numero di blocchi minati successivamente ad esso.

Il blocco genesis

Il primo blocco della storia, il "blocco genesis", è stato minato il 3 Gennaio del 2009, contiene un'unica transazione di 0.0 BTC e ha un reward di 50 BTC. Riporta al suo interno il titolo della testata britannica "The Times" pubblicato lo stesso giorno: *"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"* 1.3. Questo primo blocco ha altezza 0.[7]



Figura 1.3: The Times del 3 Gennaio 2009

1.6 Full Node

Un Full Node è un programma che convalida transazioni e blocchi. Questi nodi completi aiutano la rete accettando le transazioni e i blocchi da altri nodi, convalidando tali transazioni e blocchi e quindi inoltrandoli ad altri nodi, sempre completi. Ogni nodo completo possiede tutti i dati di tutta la Blockchain, rendendolo un database contenente lo storico di tutte le transazioni mai avvenute e contribuendo al network peer-to-peer. Al momento l'intera Blockchain Bitcoin è contenuta in quasi 400 GB di memoria. Ogni persona interessata può scaricare il software di Bitcoin Core sul sito *bitcoin.org* [6] ed inoltre può creare e gestire un proprio wallet. Il software si sincronizzerà al *network* ad ogni avvio.

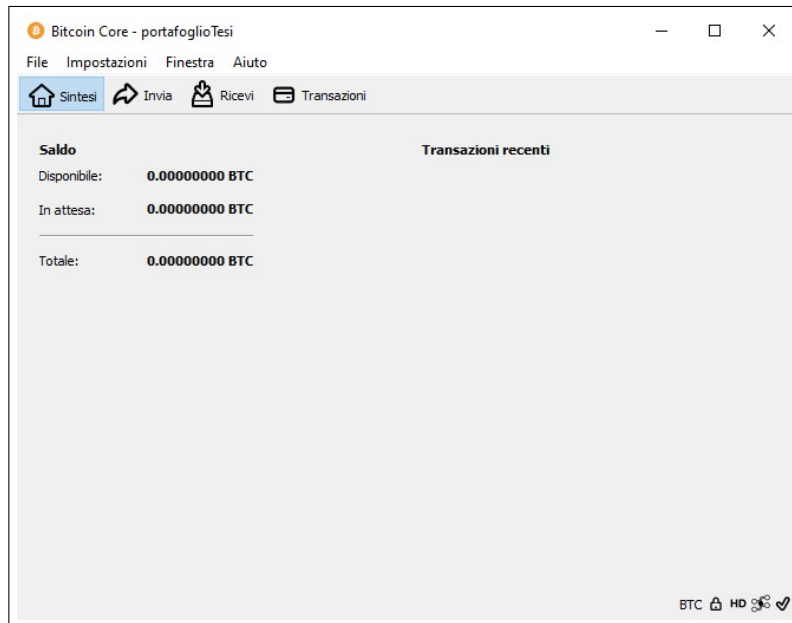


Figura 1.4: Schermata del proprio wallet all'avvio di Bitcoin Core

1.7 Mining e la Proof of Work

1.7.1 Il processo del mining

Il *mining* è il processo tramite cui le transazioni Bitcoin vengono validate e inserite nel registro pubblico, cioè nella Blockchain Bitcoin [1]. È importante capire che all'interno del network Bitcoin ci sono nodi detti Full Node (si veda la sezione 1.6) che, come già detto, gestiscono il registro pubblico e i nuovi blocchi, e nodi detti **Miner**, che in più si occupano di validare le transazioni in cambio di *fee* e della *reward* per aver minato un nuovo blocco. Prima di tutto occorre specificare la differenza tra *full node* e *miner*: i primi gestiscono e validano blocchi già minati, contententi transazioni, che a loro volta sono stati minati e validati dai minatori. Le transazioni effettuate, ma non ancora validate, vengono trasmesse in broadcast a tutti i nodi del network. I nodi *miner* si occupano di fare una prima verifica delle transazioni, controllando se superano alcuni criteri tecnici. Passata questa prima verifica, le transazioni vengono inserite in una

mempool (memory pool) e inoltrate ad altri nodi. A questo punto inizia la vera fase del *mining*: le transazioni vengono prese dalla *mempool* per essere inserite in un blocco e quindi validate. Ogni nuovo blocco contiene un'insieme di transazioni scelte dal *miner* e ognuna di esse ha n input e m output. L'unica transazione ad avere solo l'output è la transazione **Coinbase**, ovvero la transazione che lo stesso *miner* inserisce nel blocco che contiene sé stesso come destinatario e la *reward* per aver minato il blocco. Come già introdotto in 1.3.1, Bitcoin è una valuta basata su un sistema deflazionario, perché il numero massimo di bitcoin che Nakamoto ha stabilito che potranno mai esistere è 21 milioni. Al contrario dei sistemi monetari a cui siamo abituati che, essendo inflazionari, non hanno questo limite e viene immessa continuamente nuova moneta nel sistema. Dato che è tramite la transazione coinbase che si introduce nuova moneta nel sistema Bitcoin, questo si traduce nel fatto che la *reward* per aver minato un nuovo blocco non è fissa: ogni 4 anni (circa 210.000 blocchi) si attua il processo di *halving*, in cui questa *reward* viene dimezzata. Nel 2009 il valore iniziale era di 50 BTC, a seguito di diversi dimezzamenti ora il valore è di 6.25 BTC. Questo processo di dimezzamento non è infinito, infatti si arriverà al punto in cui il dimezzamento non sarà possibile, perché farlo porterebbe un valore più piccolo della più piccola unità di misura consentita nel protocollo Bitcoin, cioè $1 \text{ satoshi} = 10^{-8} \text{ BTC} = 0.00000001 \text{ BTC}$. A quel punto l'unica ricompensa per i minatori sarà rappresentata dalle *fee* delle transazioni. Ogni nuovo blocco di transazioni viene minato all'incirca ogni 10 minuti. Questo significa che in due settimane dovrebbero essere minati circa 2016 blocchi. Per mantenere stabile questo valore il protocollo Bitcoin regola la difficoltà aumentandola o diminuendola ogni due settimane.

1.7.2 Creare un blocco e la Proof of Work

Qual è il processo tramite cui un miner mina un blocco?

Come spiegato in precedenza, il nodo *miner* raccoglie un'insieme di transazioni, ognuna con n input e m output, in cui l'input corrisponde all'output di una o più transazioni non spese ricevute in precedenza (per approfondire vedere 1.9), e un'unica transazione detta **coinbase** che ha solo l'output corrispondente al *miner* stesso, e contenente la *reward* e le *fee* che gli spettano una volta minato il blocco. Il processo di *mining* consiste nella ricerca di un valore, detto *nonce*.

Le transazioni scelte vergono organizzate in una struttura dati ad albero chiamata *Merkle Tree* utilizzata spesso in crittografia. L'*header* del blocco conterrà il valore *nonce*, l'*hash del blocco precedente* e il valore della *root hash* del *Merkle Tree*. Il processo consiste nel trovare un valore *nonce* tale che concatenato al *hash del blocco precedente* e alla *root* del *Merkle Tree*, la funzione Hash risultante ha un valore minore di un certo *target*.

$$\text{Hash}(\text{nonce}|\text{hash_block_prec}|\text{MerkleRoot}) < \text{target}.$$

Il *target* è scelto dal protocollo Bitcoin in base al potere computazionale presente nel network e serve a regolare la difficoltà del processo. Infatti, ogni due settimane il valore *target* può variare in base al numero di blocchi minati. Se il numero di blocchi minati è stato più di uno ogni 10 minuti, questo valore *target* sarà diminuito, aumentando così la difficoltà di trovare un valore *nonce* adeguato, altrimenti sarà aumentato per semplificare il processo di *mining*. Questo processo è chiamato Proof of Work. La ricerca di questo *nonce* è un processo elaborato che richiede potenza di calcolo. Satoshi stesso, in *Bitcoin: A Peer-to-Peer Electronic Cash System* [14], parla di spendere potenza CPU (Central Processing Unit) e consumare energia elettrica,

al fine di aumentare la quantità di moneta nel sistema, paragonando il *mining* al processo di estrazione dei minatori d'oro, che spendono risorse per incrementare la quantità di oro in circolazione.

1.7.3 Mining Farm

In un primo momento, per il *mining* venivano utilizzate le CPU. Ci volle poco a capire che le Graphics Processing Unit (GPU), ovvero le schede grafiche, erano in grado di migliorare le prestazioni, essendo in grado di svolgere calcoli in parallelo, dando modo alla scheda di lavorare in contemporanea su range di *nonce* diversi. Pochi anni dopo vennero utilizzate le FPGA (*field-programmable gate array*), delle schede programmate per svolgere funzioni di *hashing*. Attualmente esistono delle schede costruite appositamente per lo scopo, le ASIC (application-specific integrated circuit). La crescita del valore del bitcoin ha fatto sì che si generasse una ricerca all'oro che si è tradotta nella costruzione delle Mining Farm, cioè degli edifici costruiti al fine di contenere macchine con nodi Miner in esecuzione. Considerando che il processo di *mining* consuma molta potenza di calcolo ed elettricità, è vantaggioso posizionare queste Mining Farm in paesi dove la corrente elettrica è economica e in cui il clima è freddo, per poter risparmiare sul processo di raffreddamento dell'*hardware* in uso. Un esempio virtuoso è rappresentato dalla cittadina canadese Ocean Falls che si è ripresa dalla crisi economica causata dalla chiusura delle sue miniere, sfruttando l'energia elettrica non utilizzata, per alimentare una *farm* contenente 530 *computer*. [2]

1.7.4 Problema dei generali bizantini

Il problema dei generali bizantini nasce come variante del problema dei generali cinesi, un problema dell'**informatica distribuita** in cui due generali devono raggiungere

un accordo comune sull'attacco o sulla ritirata, ma l'unico modo di comunicare che hanno è tramite messaggeri che potrebbero non arrivare mai.

Leslie Lamport "ruba" questo problema, ponendo il problema nei termini di un gruppo di generali, alcuni dei quali possono essere traditori, che devono raggiungere una decisione comune. Il nome "bizantini" è stato scelto per non offendere nessun lettore. L'idea del problema è quella di raggiungere il consenso in circostanze in cui potrebbero essere presenti errori e informazioni discordanti, comunicando solo tramite messaggi [12].

Immaginiamo l'esercito Bizantino organizzato in divisioni, ognuna capeggiata da un generale. Questi ultimi devono mettersi d'accordo sul da farsi, attaccare o ritirarsi. Inoltre, ogni generale può essere Leale o un Traditore che cerca di impedire il raggiungimento di una decisione comune. I traditori potrebbero inviare informazioni errate o comportarsi in modo arbitrario. Se alcuni membri del gruppo hanno ricevuto una informazione, mentre altri ne hanno ricevuto una differente, i membri non agiranno all'unisono, facendo sì che il gruppo sia incapace di coordinare in modo efficace le proprie azioni.

Nel network Bitcoin

In informatica, un ambiente distribuito, come quello della rete Bitcoin, deve essere progettato in modo tale da risolvere il problema dei generali bizantini per poter essere affidabile, cioè deve avere la proprietà chiamata Byzantine Fault Tolerance (BTF). Satoshi Nakamoto ha risolto il problema con il 51% attack, come spiegato nel paragrafo successivo.

1.7.5 51% attack

Tornando al *mining*, il protocollo Bitcoin è pensato per proteggersi da eventuali *miner* "traditori", interessati a falsificare un blocco e le transazioni all'interno di esso. Nakamoto ha risolto il problema progettando il protocollo in modo tale che non sia la maggioranza dei nodi a vincere, altrimenti si potrebbero generare dei nodi falsi, che contengono un blocco falso, ma è la potenza di calcolo all'interno del network che vince. Questo significa che attaccare Bitcoin è possibile possedendo il 51% della potenza di calcolo ma non è vantaggioso perché sarebbe più costoso di quanto si guadagnerebbe in termini economici. La regola del 51% rende Bitcoin teoricamente attaccabile ma non nella pratica e nel buonsenso.

1.8 Breve introduzione alle Transazioni

Una transazione è il trasferimento di denaro da un indirizzo A ad un indirizzo B. È un messaggio in cui l'indirizzo A (mittente) inserisce il quantitativo di bitcoin che vuole inviare e la chiave pubblica di B (destinatario), il tutto viene firmato con la chiave privata di A. Questo messaggio viene inoltrato in broadcast all'intero network Bitcoin e sarà inserito in un blocco e validato dai minatori come spiegato in 1.7.

La prima transazione commerciale

La prima transazione commerciale della storia di Bitcoin, cioè la prima transazione in cui è stato acquistato qualcosa di reale, è avvenuta il 22 Maggio 2010. Un uomo di nome Laszlo Hanyecz scrisse su Bitcoin.org che era interessato a spendere 10.000 BTC in suo possesso per due pizze. Quattro giorni dopo un altro utente gli ordinò le pizze, prendendosi in cambio i bitcoin. Quel 22 Maggio viene ricordato ogni anno e festeggiato come Bitcoin Pizza Day.([13])

1.9 Modello UTXO

Il protocollo Bitcoin utilizza il modello UTXO (*unspent transaction output*) per la gestione delle transazioni [1]. UTXO rappresenta gli output delle transazioni precedenti non spesi. Siamo abituati a considerare i nostri soldi con il modello Account Based Model, il classico modello delle Banche, in cui abbiamo il nostro saldo complessivo da cui vengono decurtati i soldi che vogliamo spendere. Il modello UTXO funziona in modo diverso: le UTXO sono indivisibili, infatti l'input di una transazione è formato da più *unspent* che messe insieme² formano la quantità di moneta che si vuole trasmettere ad un altro indirizzo Bitcoin. Qualora l'ammontare delle UTXO in input superi la quantità di bitcoin che si vuole trasmettere, si genererà una ulteriore transazione contenente il "resto", che ha come destinatario il mittente della transazione. Il resto gli sarà accreditato su un nuovo indirizzo Bitcoin, per aumentare il livello di sicurezza.

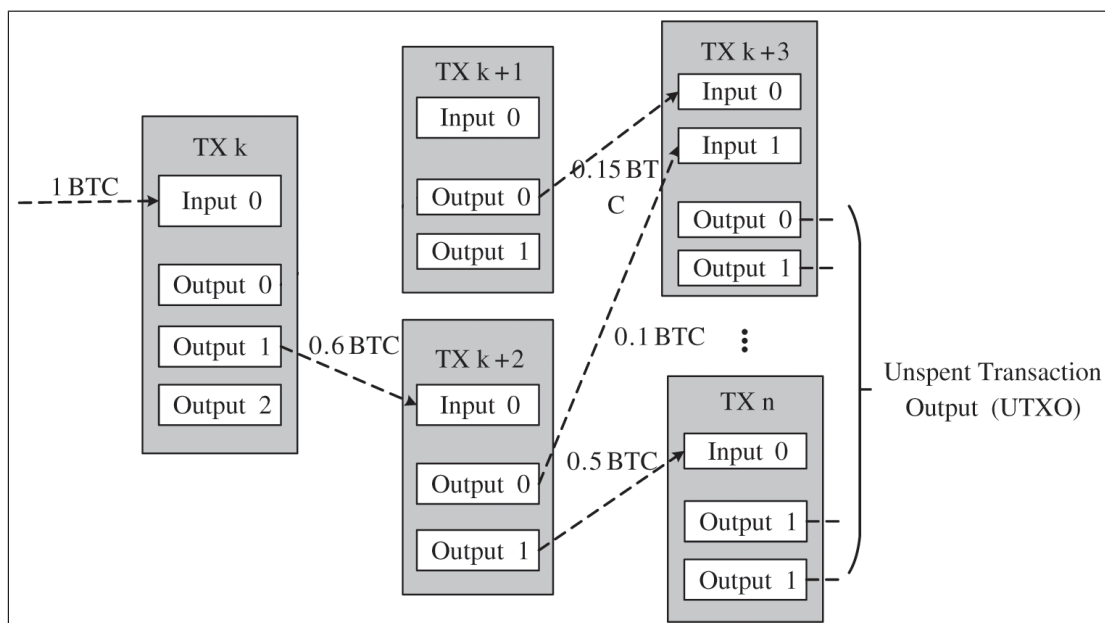


Figura 1.5: Modello UTXO

²L'unica eccezione è la Coinbase.

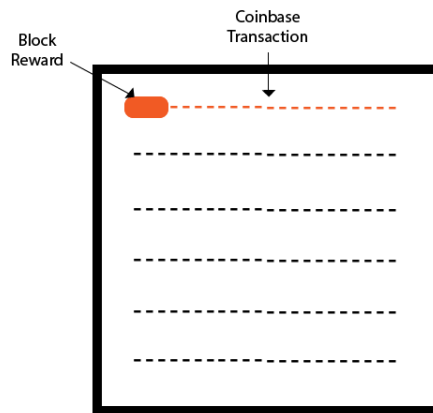


Figura 1.6: Transazione Coinbase con *reward*, l'unica transazione ad avere solo l'Output. Sono i nuovi *coin* generati

1.9.1 Struttura di una Transazione

Una transazione è costituita da n Input e m Output. Il mittente mette nell'input le transazioni che a sua volta ha ricevuto in precedenza, le UTXO, da cui prendiamo i soldi da spendere. Nell'output verranno inseriti i valori in BTC e i rispettivi indirizzi a cui inviarli. Il *locktime* indica l'altezza minima del blocco entro il quale la transazione sarà minata.

<u>Input</u>	<u>Output</u>
prev tx id index scriptSig	value scriptPubKey
locktime	

Figura 1.7: Struttura di una transazione

Transazione: Input

Come detto in precedenza, l'Input deve far riferimento a dei *coin* ricevuti in altre transazioni, di conseguenza il primo campo dell'Input raccoglie i riferimenti a queste transazioni precedenti, tramite il loro *hash*. Poi ci sono i relativi *index* per indicare quale *unspent* della transazione precedente si vuole spendere e la *signature*, cioè la *scriptSig*. Tramite quest'ultima il mittente firma la transazione e dimostra che i *coin* sono in suo possesso.

Transazione: Output

Nella parte dell'Output deve essere contenuto i valori che si vogliono trasferire e l'*address* di ogni destinatario.

1.10 Fee delle Transazioni

Una volta compiuta una transazione, l'utente deve aspettare un intervallo di tempo variabile prima che la sua transazione venga inserita in un blocco e quindi convalidata. Finché questo non avviene il destinatario non può spendere a sua volta i *coin* ricevuti. L'intervallo di tempo dal momento della formulazione della transazione alla sua effettiva convalida dipende molto dalle *fee* che l'utente è disposto a pagare al Miner, che spenderà energia elettrica per inserirla nel blocco e prendersi la *reward* che gli spetta. Come spiegato in 1.7, la *reward* che riceverà il minatore è formata dal premio per aver minato un blocco e, appunto, da queste *fee*. Può succedere che transazioni con *fee* esigue vengano sorpassate da transazioni con *fee* sostanziose. In più occorre tener conto che questa "mancia" al minatore la si paga per ogni *unspent* in Input, e non per la transazione in totale.

Capitolo 2

Strumenti per l'estrazione dei dati nella Blockchain di Bitcoin

In questo capitolo verranno esposti gli strumenti utilizzati e le modalità di estrazione dei dati.

2.1 Strumento utilizzato: Rusty-blockparser

Come spiegato nella Sezione 1.6, è possibile diventare un nodo Bitcoin scaricando nel proprio computer il software Bitcoin Core. Così facendo non solo si contribuirà al progetto, ma si avrà una copia dell'intera Blockchain di Bitcoin nel proprio nodo. I dati saranno in alcuni *file* **blk<numero>.dat**. A questo punto serve un Parser, cioè un *software* in grado di interpretare i dati e "tradurli" in un altro linguaggio. Per questo caso è stato scelto **Rusty-blockparser**, un *parser* per la Blockchain di Bitcoin (ma anche altre cryptovalute) scritto in linguaggio **Rust**. È stato possibile trovarlo e scaricarlo tramite Github. Per sfruttare le sue potenzialità è stato usato Git Bash.

[16]

```
scrib@PC-Majestic MINGW64 ~
$ rusty-blockparser -h
Multithreaded Blockchain Parser written in Rust 0.8.1
gcarq <egger.m@protonmail.com>

USAGE:
    rusty-blockparser.exe [FLAGS] [OPTIONS] [SUBCOMMAND]

FLAGS:
    -h, --help            Prints help information
    -V, --version          Prints version information
    -v                    Increases verbosity level. Info=0, Debug=1, Trace=2 (default: 0)
    --verify              Verifies the leveldb index integrity and verifies merkle roots

OPTIONS:
    -d, --blockchain-dir <blockchain-dir> Sets blockchain directory which contains blk.dat files (default:
                                           ~/.bitcoin/blocks)
    -c, --coin <NAME>                  Specify blockchain coin (default: bitcoin) [possible values: bitcoin,
                                           testnet3, namecoin, litecoin, dogecoin, myriadcoin, unobtanium]
    -e, --end <NUMBER>                Specify last block for parsing (inclusive) (default: all known blocks)
    -s, --start <NUMBER>              Specify starting block for parsing (inclusive)

SUBCOMMANDS:
    balances      Dumps all addresses with non-zero balance to CSV file
    csvdump       Dumps the whole blockchain into CSV files
    help         Prints this message or the help of the given subcommand(s)
    simplestats   Shows various Blockchain stats
    unspentcsvdump Dumps the unspent outputs to CSV file
```

Figura 2.1: Potenzialità di rusty-blockparser

Rusty-blockparser è in grado di sfogliare i blocchi ed estrapolare i dati che inserirà in *file csv* (*comma-separated values*). I *file csv* sono semplici da utilizzare per importare o estrapolare dati, perchè sono organizzati e separati da virgole. Ad esempio, tramite EXCEL, noto *software* per l'analisi dati, è possibile aprire questi file e trovare i dati già organizzati in una tabella.

2.1.1 File csv

Rusty-blockparser permette tre comandi:

- **balances:** permette di ottenere un *file csv* *balances.csv* contenente gli indirizzi Bitcoin con il relativo bilancio. Con il seguente formato:

address ; balance

- **unspentcsvdump:** permette di ottenere un *file csv* *unspent.csv* contenente le UTXO associate ad ogni indirizzo Bitcoin.
Nel seguente formato:

txid ; indexOut ; height ; value ; address
--
- **csvdump:** permette di ottenere quattro *file csv*:

- *blocks.csv*, nel formato:

block_hash ; height ; version ; blocksize ; hashPrev ; hashMerkleRoot ; nTime ; nBits ; nNonce

- *transactions.csv*, nel formato: txid ; hashBlock ; version ; lockTime

- *tx_in.csv*, nel formato: txid ; hashPrevOut ; indexPrevOut ; scriptSig ; sequence

- *tx_out.csv*, nel formato: txid ; indexOut ; height ; value ; scriptPubKey ; address

balances
balances.csv
address ; balance
unspentcsvdump
unspent.csv
txid ; indexOut ; height ; value ; address
csvdump
blocks.csv
block_hash ; height ; version ; blocksize ; hashPrev ; hashMerkleRoot ; nTime ; nBits ; nNonce
transactions.csv
txid ; hashBlock ; version ; lockTime
tx_in.csv
txid ; hashPrevOut ; indexPrevOut ; scriptSig ; sequence
tx_out.csv
txid ; indexOut ; height ; value ; scriptPubKey ; address

Figura 2.2: Tabella riassuntiva dei comandi e dei *file* generabili di Rusty-blockparser

2.1.2 Gestione e generazione file

Già da una prima generazione dei *file* risultano evidenti alcune problematiche legate allo spazio di memoria e al tempo di calcolo dei suddetti *file*. Si poteva agire in due modi diversi:

- **Generare i file a tempo di esecuzione.** Python possiede una libreria chiamata **subprocess** che, come spiegato nella Sezione 2.2, permette di eseguire sottoprocessi richiamati tramite il codice Python in esecuzione. Questo significa

che è stato possibile richiamare, tramite Git, **Rusty-blockparser** direttamente dal codice, per generare *file* e aprirli a tempo di esecuzione nel codice stesso. Questa opzione si è ritenuta necessaria nelle analisi che richiedevano di analizzare singoli blocchi (o intervalli di blocchi) che venivano richiamati tramite un ciclo *for*. Una volta lette le informazioni, venivano sovrascritti dal *file* del blocco successivo. Questa scelta rendeva il computer inutilizzabile per qualsiasi altra attività vista la comparsa delle finestre del sottoprocesso (circa una al secondo) in esecuzione al centro del desktop.

- **Generare tutti i file prima.** Generando i *file* necessari prima, si sarebbe creata nella propria macchina un'ottima fonte da cui trarre dati, senza perdere tempo a ricalcolare i *file* ogni volta. È, a parer mio, la scelta più efficiente per alcune analisi, se si dispone di un disco spazioso e una buona capacità di organizzazione. Difatti ogni tipologia di *file* aveva dimensioni diverse a seconda delle informazioni contenute e a seconda dell'intervallo di blocchi preso in esame. Calcolare ogni volta i *file*, sarebbe significato non solo metterci di più a livello di tempo, ma si sarebbe rischiato anche di perdere ore di calcolo per colpa un errore banale nel codice. Ogni comando è stato diviso in sottointervalli di blocchi per poter essere generato rispettando i limiti di memoria RAM (*random access memory*) a disposizione. L'esempio più eclatante è stato quello dei *file transaction.csv*.

Nome	Dimensione
transactions-0_214553	1.383.357 KB
transactions-214554_272422	2.319.297 KB
transactions-272423_330290	3.045.984 KB
transactions-330291_388158	6.014.131 KB
transactions-388159_446026	11.379.068 ...
transactions-446027_473988	6.948.726 KB
transactions-473989_501950	6.769.582 KB
transactions-501951_515542	2.549.765 KB
transactions-515543_529133	2.265.028 KB
transactions-529134_542730	2.561.191 KB
transactions-542731_556315	3.334.120 KB
transactions-556316_569916	3.800.674 KB
transactions-569917_583500	4.345.472 KB
transactions-583501_597401	3.931.133 KB
transactions-597402_610680	3.738.245 KB
transactions-610681_625310	4.044.972 KB
transactions-625311_639942	4.015.198 KB
transactions-639943_654575	4.178.497 KB
transactions-654576_669210	4.133.698 KB
transactions-683846_698461	3.192.264 KB

Figura 2.3: *File transactions* generati per le analisi svolte in questo lavoro.

Ogni *file* è stato organizzato con: `<nome_file-intervallo_blocchi>.csv`

2.1.3 Difficoltà e limiti riscontrati

Come già accennato, alcuni comandi hanno richiesto più suddivisione dei blocchi per far sì che i *file* non fossero troppo grandi per essere analizzati dalla RAM a disposizione. Per alcune tipologie di comando questo non è bastato. I *file tx_in.csv*, generati tramite il comando **csvdump**, richiedevano uno spazio di memoria tale da non renderli generati e letti tutti. Ad esempio, un *file tx_in.csv* calcolato su diecimila blocchi minati in questi ultimi anni, pesa almeno 20 GB. Questo li rende impossibili da analizzare tutti per via dei mezzi a disposizione. Una problematica di Rusty-blockparser è che, del comando **csvdump**, non permette di calcolare solo alcuni *file*, ma costringe a calcolarli tutti. Questo comporta che se si vuole avere il *file blocks.csv* su un intervallo grande di blocchi, si deve avere a disposizione abbastanza spazio e

tempo da dedicare anche ai *file tx_in.csv*, che saranno poi cancellati. Inoltre, più si va avanti nella Blockchain Bitcoin, più le informazioni per singolo blocco aumentano, facendo sì che, per esempio, un *file* su diecimila blocchi calcolato negli ultimi anni, fosse molto più pesante dello stesso intervallo di blocchi calcolato nei primi anni di vita del bitcoin.

Senza scendere troppo nei dettagli, i *file csv* utilizzati in questa tesi hanno superato i 400 GB di memoria necessaria. Inoltre, un *full node* Bitcoin, come spiegato nella Sezione 1.6, pesa quasi 400 GB.

2.2 Analisi file csv con codice Python

Una volta ottenuti i *file csv*, bisognava analizzarli. È stato utilizzato Python ed una sua libreria: **pandas**. [15] Quest'ultima non è stata l'unica libreria Python utilizzata:

- **glob**: un modulo che fa già parte della libreria standard di Python ed è utile per estrapolare tutti i *file csv* dalla *directory*, per poter così evitare l'inserimento manuale nel codice. Utile in questi casi in cui si analizzano moli di *file* e dati considerevoli.
- **Matplotlib**: per disegnare eventuali grafici e rendere di facile lettura i risultati ottenuti.
- **datetime**: un modulo che permette di analizzare e manipolare le date.
- **subprocess**: una libreria che permette di eseguire sottoprocessi direttamente dal codice Python.
- **json**: una libreria che permette, tramite la funzione *dump()*, di convertire oggetti nel rispettivo oggetto *json*.

Non importa quale sia il tuo problema, sicuramente esiste una libreria Python che può risolverlo!

2.3 Macchina utilizzata

Il computer che è stato utilizzato per le analisi ha a disposizione:

- CPU: AMD Ryzen 5 1400 3.2GHz.
- RAM: Corsair Vengeance DDR4, 3200 MHz, C16, 16 GB.

Capitolo 3

Analisi dati sulla Blockchain di Bitcoin

In questo capitolo verranno approfondite le analisi fatte nella Blockchain di Bitcoin, i prototipi sviluppati e i limiti computazionali riscontrati, oltre cui non si è potuto procedere.

3.1 Periodi di interesse

Rusty-blockparser permette di eseguire il *parsing* su dei blocchi precisi, decisi dall'utente e specificabili prima di inviare il comando nell'interfaccia di Git. Per questo progetto si è deciso di analizzare i blocchi dividendoli in base al periodo storico a cui appartengono, perché ritengo che sia più interessante analizzare i cambiamenti nel corso del tempo, che ottenere un calderone di dati senza poterli contestualizzare. In ogni caso questa divisione sarebbe stata una scelta obbligata per motivi di complessità computazionale, risultava impossibile infatti, dati i mezzi a disposizione, analizzare la Blockchain utilizzando un unico *file csv*.

3.1.1 Divisione temporale dei blocchi

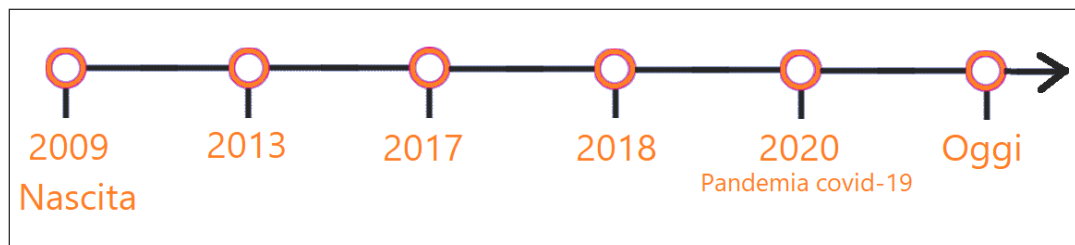


Figura 3.1: Divisione temporale

Primi blocchi: nascita

Una prima divisione dei blocchi è stata fatta dalla nascita, quindi dal blocco 0, fino al blocco 214553, l'ultimo blocco minato il 31 Dicembre 2012.

2013-2016

Tra il Gennaio 2013 e il Dicembre 2016, il bitcoin ha continuato a crescere lentamente di valore e ha continuato ad affermarsi. L'analisi dei blocchi è stata fatta dal blocco 214554, minato a mezzanotte del 1 Gennaio 2013, fino al blocco 446026, minato il 31 Dicembre 2016

2017

Ho deciso di analizzare l'anno 2017, perché è in questo anno che osserviamo la prima vera affermazione e crescita del bitcoin. In pochi mesi il suo valore cresce esponenzialmente, fino ad arrivare a valere quasi 20.000 dollari. L'analisi è stata fatta dal blocco 446027 fino al blocco 501950.

Discesa: 2018-2019

La crescita improvvisa del 2017 si è rivelata una "bolla", difatti dal 2018 al 2019 il suo valore è calato drasticamente fino al momento in cui un bitcoin valeva poco più

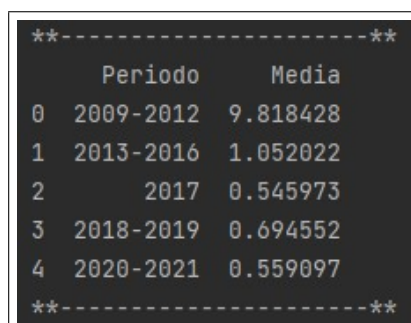
di 3.000 dollari. L'analisi è stata fatta dal blocco 501951 fino al blocco 610680.

Ulteriore crescita: pandemia 2020

A seguito della pandemia di Covid-19, si è osservata una nuova crescita esponenziale del valore del bitcoin. Stavolta il picco raggiunto è stato di quasi 65.000 dollari. L'analisi dei blocchi è stata fatta dal blocco 610681, minato ad inizio 2019, fino al blocco 698461, minato il 31 Agosto 2021.

3.2 Analisi dei balance degli utenti

La prima analisi svolta è sui file *balances.csv* nel formato: `address ; balance`. Viene calcolata la media di bitcoin posseduti, inizialmente nei periodi di interesse (3.1). Nelle analisi non è stata tenuta in considerazione l'ipotesi di chi ha più *address*, non potendolo verificare. Tuttavia non bisogna dimenticare che molti utenti ne hanno di più per aumentare la sicurezza.



	Periodo	Media
0	2009-2012	9.818428
1	2013-2016	1.052022
2	2017	0.545973
3	2018-2019	0.694552
4	2020-2021	0.559097

Figura 3.2: Bilancio degli address

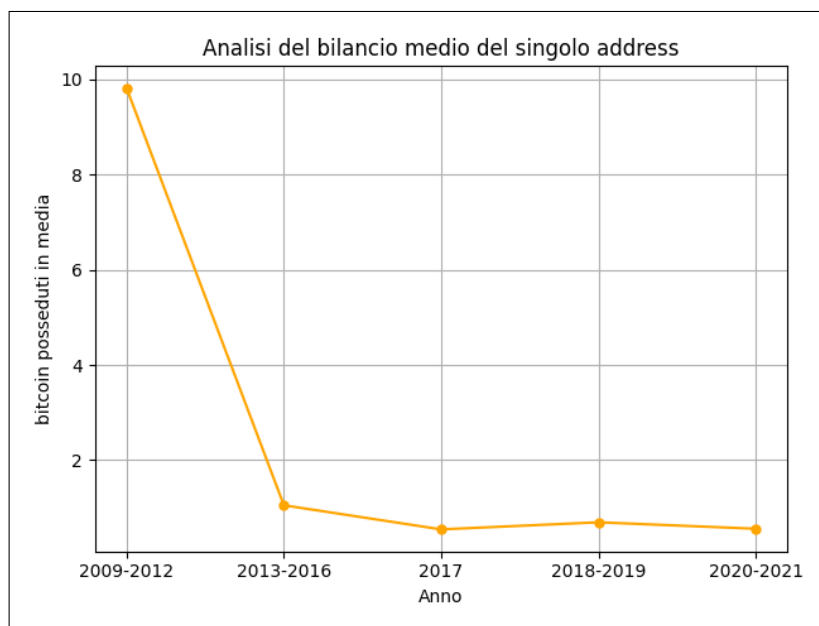


Figura 3.3: Grafico dell'andamento del bilancio degli indirizzi

Risulta evidente come la decrescita del valore medio di bitcoin posseduti sia calato in parallelo alla crescita del suo valore. Notare come, seppur di poco, il bilancio medio sia risalito nel periodo 2018-2019.

3.2.1 Balance complessivi

A questo punto si è deciso di analizzare tutti gli *address* Bitcoin esistenti.¹

È stato lanciato il comando:

```
rusty-blockparser -d <percorso_file_bitcoin> s 0 e 698461 balances <percorso_file_destinazione>
```

ed è stato calcolato un *file* complessivo di tutti gli *address* Bitcoin esistenti fino al 31 Agosto 2021, con il relativo *address*. Il *file* è stato calcolato in circa 24 ore e pesa 1.4 GB. È stato anche scritto un codice Python per unire i *file address* già a disposizione, evitando di ricalcolarli, ma dopo diverse ore di esecuzione è stato evidente che non si sarebbe risparmiato tempo e anzi, si sarebbe potuti andare incontro ad errori nel *file*

¹Rusty-blockparser prende solo gli *address* con un bilancio superiore a 0 BTC.

per via di possibili errori di distrazione nel codice.

Non sono state fatte ulteriori analisi su questo *file* perché non ne sarebbe valsa la pena. Ad una prima analisi il *file* è risultato contenere 32881413 *address*. Per analisi di *routine*, come la ricerca dei bitcoin finora posseduti in totale e in media, ci sarebbero voluti giorni, infatti ad un primo test la somma di diecimila *value* di *address* ha impiegato un'ora per essere calcolata.

3.3 Analisi del valore delle UTXO

Sul file *unspent.csv*, contenente le UTXO associate ad ogni indirizzo Bitcoin e nel formato: `txid ; indexOut ; height ; value ; address`, si è deciso di mettere in evidenza il valore medio delle UTXO e della relativa altezza (in media) del blocco a cui appartengono, utilizzando i campi *height* e *value*.

	Periodo storico	Altezza blockchain	Media valore UTXO
0	2009-2012		107276.5 0.328145
1	2013-2016		330290.0 0.030591
2	2017		473988.5 0.031896
3	2018-2019		556315.5 0.030638
4	2020-2021		654571.5 0.033221

Figura 3.4: Analisi UTXO

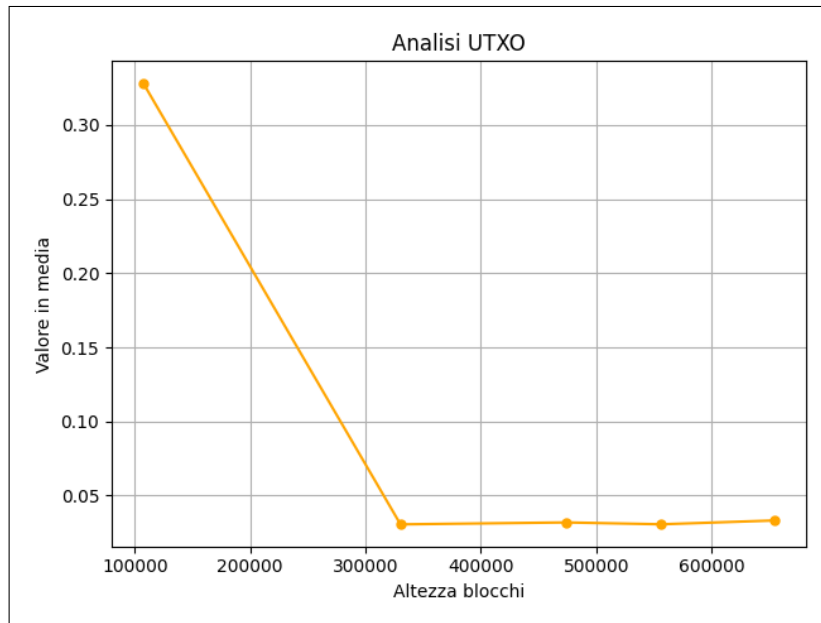


Figura 3.5: Grafico analisi UTXO

3.4 Analisi balance e UTXO

Si è deciso di incrociare i dati derivati dall'analisi del bilancio degli utenti (3.4) e i dati derivabili dal *file unspent.csv*, questa volta utilizzando i campi *value* e *address*. È possibile notare come il valore medio delle UTXO sia rimasto pressoché invariato, al contrario del bilancio totale che ha osservato un clamoroso picco tra il 2009 e il 2012, per poi diminuire, parallelamente alla crescita del valore del bitcoin.

	Periodo	Valore medio balance	Valore medio UTXO
0	2009-2012	9.818428	0.328145
1	2013-2016	1.052022	0.030591
2	2017	0.545973	0.031896
3	2018-2019	0.694552	0.030638
4	2020-2021	0.559097	0.033221

Figura 3.6: Analisi contenente il balance medio e il valore medio delle UTXO

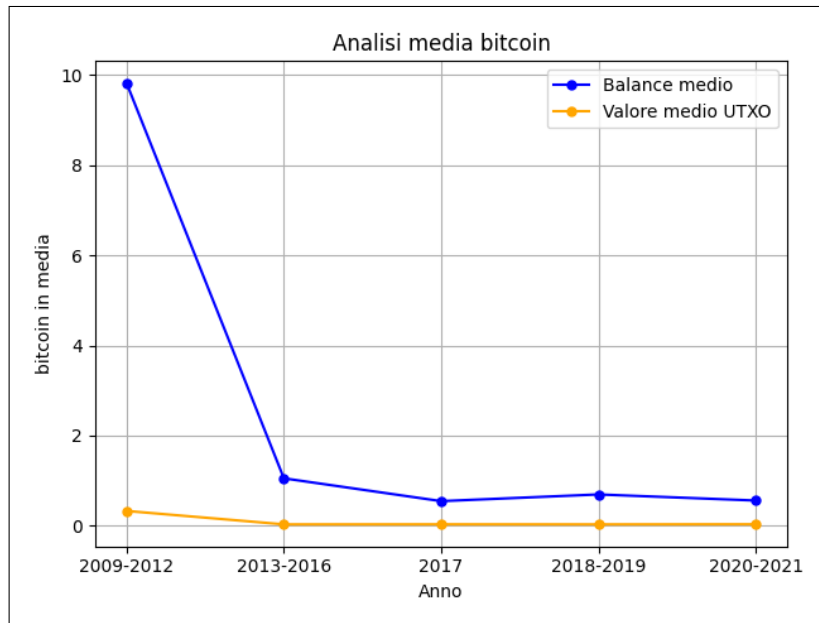


Figura 3.7: Grafico contenente il balance medio e il valore medio delle UTXO

3.5 Analisi Blocksize

Si è deciso di mettere a confronto la variazione delle *size* dei blocchi, mano a mano che bitcoin iniziava ad essere usato sempre di più. Chiaramente è stata riscontrata una crescita evidente. Prima però occorre approfondire da cosa è dovuta la *size* di ogni blocco.

3.5.1 Blocksize

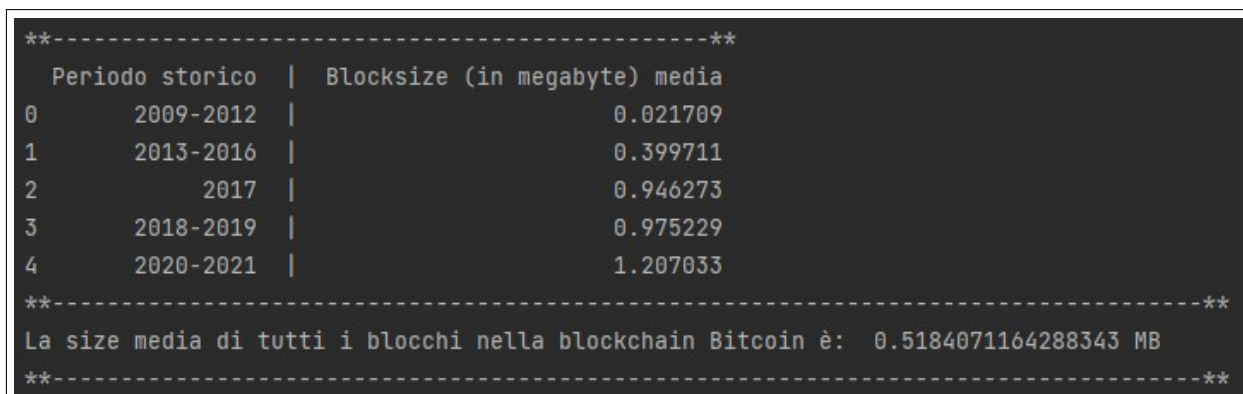
Nel protocollo Bitcoin, Satoshi Nakamoto ha fissato il limite di dimensione del blocco ad 1 megabyte (= 1,048,576 Bytes). Non ha mai specificato perché ha aggiunto questo limite, si possono solo formulare ipotesi. È stato ipotizzato che fosse una misura anti-spam, per evitare che un utente malintenzionato possa sovraccaricare la rete Bitcoin con blocchi pieni di transazioni false o che fosse solo una misura temporanea, ma in

questo secondo caso non si sa quando avesse previsto un suo aumento o la revoca. ([10])

3.5.2 Risultati analisi

Una volta chiarito e contestualizzato cosa si sta analizzando, si è in grado di capire meglio i seguenti risultati. Questi risultati non sono sorprendenti, è chiaro che con la crescita del valore della valuta, sia cresciuto il traffico di transazioni da validare.

Come si può notare, nel periodo 2020-2021 la blocksize sembra sfiorare il limite imposto a 1 MB. Non è corretto: nel limite di 1 MB non sono comprese le *signature* necessarie per ogni transazione e altri dati che non saranno approfonditi in questo lavoro.



```

**-----**
Periodo storico | Blocksize (in megabyte) media
0      2009-2012 | 0.021709
1      2013-2016 | 0.399711
2      2017      | 0.946273
3      2018-2019 | 0.975229
4      2020-2021 | 1.207033
**-----**
La size media di tutti i blocchi nella blockchain Bitcoin è: 0.5184071164288343 MB
**-----**

```

	Periodo storico	Blocksize (in megabyte) media
0	2009-2012	0.021709
1	2013-2016	0.399711
2	2017	0.946273
3	2018-2019	0.975229
4	2020-2021	1.207033

La size media di tutti i blocchi nella blockchain Bitcoin è: 0.5184071164288343 MB

Figura 3.8: Analisi blocksize

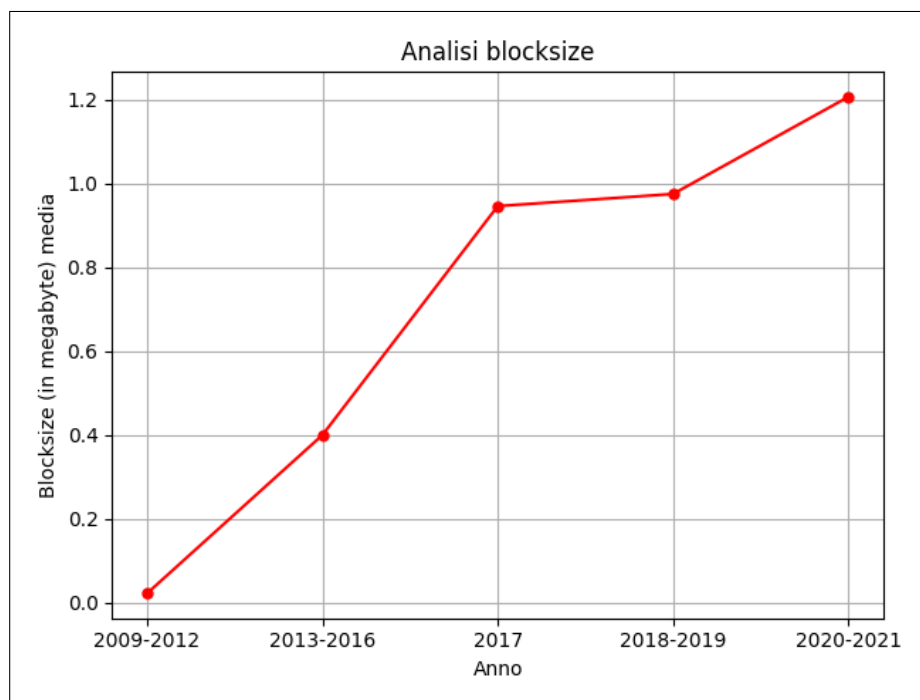


Figura 3.9: Grafico analisi blocksize

3.6 Analisi Transazioni nei blocchi

Il file *transactions.csv* è nel formato: `txid ; hashBlock ; version ; lockTime`. Da esso si è deciso di contare quante transazioni c'erano in ogni blocco (contando le ripetizioni dello stesso *hashBlock*) e di estrapolare l'*hashBlock* del blocco con più transazioni. Una volta ottenuto si è cercato nei file *blocks.csv* il momento esatto in cui tale blocco è stato minato. L'orario nei blocchi della Blockchain di Bitcoin è espresso in Unix Timestamp. Con il modulo *datetime* è stato possibile convertirlo in una "human-readable date", dando modo di leggere che il blocco contenente più transazioni nella storia di bitcoin è stato minato il 1 Agosto del 2015.

Sappiamo che i blocchi vengono minati ad intervalli di 10 minuti circa. Consultando *blockchain.com*, nella sezione *explorer*, possiamo agevolmente notare che il blocco precedente, il 367852, è stato minato 26 minuti prima e contiene solo 1.037 transazioni.

serie di chiamate di Rusty-blockparser, per ogni blocco, finché non si fosse riscontrata nuovamente la *unspent* nell'**input di una nuova trasazione**.

È possibile analizzare l'input con il file *tx_in.csv*, nel formato:

txid ; hashPrevOut ; indexPrevOut ; scriptSig ; sequence, ottenuto con il comando **csvdump** di Rusty-blockparser. Calcolando questo file per un singolo blocco, si ottiene la lista di tutte le UTXO utilizzare per formare gli importi di tutte transazioni del blocco in esame. Questo procedimento, che ha il vantaggio di generare *file* riguardanti il singolo blocco, è inattuabile per una questione di complessità computazionale. Infatti se, con un'approssimazione ottimistica, ipotizziamo che una chiamata di Rusty-blockparser impieghi solo un secondo per generare il *file* necessario, sarebbero necessarie 700.000 chiamate nel caso peggiore, ovvero 700.000 secondi (≈ 200 ore). La ricerca lineare ha, nel caso peggiore, complessità temporale $O(n)$. Questo senza considerare il tempo per scansionare ogni *file* generato.

Calcolare prima i *file* non è una soluzione per due motivi: prima di tutto la grandezza dei *file tx_in.csv* non rende possibile tenerli in memoria tutti con i mezzi a disposizione, inoltre la scansione di ogni *file* avrebbe comunque una complessità temporale che rende irrisolvibile l'analisi in questa modalità. Con una ricerca lineare quest'analisi non è attuabile. Si è allora pensato di affrontare quest'analisi con una Ricerca Binaria.

3.7.1 Ricerca Binaria

Per semplicità vediamo la Ricerca Binaria applicata ad un *array* di dimensione L , i cui estremi sono i valori i e j . Tramite questa ricerca si vuole trovare l'elemento x . Si parte con l'**assunzione** che l'*array* sia composto di valori ordinati in senso crescente. Si calcola $m = \left\lfloor \frac{i+j}{2} \right\rfloor$. A questo punto tramite degli **if** si decide come procedere. Se il valore $L[m]$ corrisponde all'elemento x ricercato, l'algoritmo restituisce m , ovvero

la locazione dell'elemento ricercato nell'*array*. Se il valore $L[m]$ è più grande di x , vorrà dire che la ricerca continuerà nella prima metà dell'*array*, altrimenti continuerà nella seconda parte, sempre considerando che stiamo lavorando sotto l'assunzione che l'*array* è ordinato.

La Ricerca Binaria ha, nel caso peggiore, complessità temporale: $O(\log n)$

```

algoritmo RicercaBinariaRic(array  $L$ , elem  $x$ , int  $i$ , int  $j$ ) --> intero
1.  if ( $i > j$ ) then return -1
2.   $m = \lfloor (i+j)/2 \rfloor$ 
3.  if ( $L[m] = x$ ) then return  $m$ 
4.  if ( $L[m] > x$ ) then return RicercaBinariaRic( $L, x, i, m-1$ )
5.  else return RicercaBinariaRic( $L, x, m+1, j$ )
    
```

Figura 3.11: Pseudocodice Ricerca Binaria

3.7.2 Nella Blockchain Bitcoin

Nella Blockchain Bitcoin, i blocchi sono ordinati in senso crescente, perciò si può applicare questo tipo di ricerca anche nella Blockchain. Sempre prendendo un campione di UTXO, l'idea è stata di ricercare ogni *unspent* tramite la ricerca binaria, per capire se e quando fosse stata rispesa.

- **Prima idea:** ricerca binaria nell'intera Blockchain Bitcoin. Chiaramente con questa tipologia di analisi i tempi necessari sono diventati più praticabili, infatti ad ogni chiamata di Rusty-blockparser si sarebbe dimezzato il numero di blocchi da analizzare alla ricerca della *unspent*. Eppure non si è risolto del tutto il problema. Infatti, bisogna considerare che la prima chiamata di Rusty-blockparser con il comando **unspentsvdump** deve generare un file comprensivo di tutte le *unspent* dal blocco 0 al blocco $m = \left\lfloor \frac{0 + 698461}{2} \right\rfloor = 349230$. Per generare questo *file* ci vogliono ore.

- **Seconda idea:** riutilizzare i *file.csv* generati in precedenza per altre analisi (3.3). Prima di implementare la ricerca binaria sulla Blockchain Bitcoin, si voleva restringere l'intervallo di blocchi su cui effettuarla, ricercando la *unspent* nei *file* già in possesso tramite una ricerca lineare. Se in un *file* veniva trovata, ma in quello successivo non era presente, significava che era stata rispesa in precedenza e quindi l'intervallo di blocchi si restringeva all'intervallo di blocchi su cui era stato calcolato il *file unspent*. Qualora la *unspent* fosse stata trovata in ogni *file* non sarebbe stata ricercata ulteriormente, immaginando che non fosse stata rispesa.

Tramite questa strategia l'intervallo di blocchi, nel caso peggiore, su cui sarebbe stato necessario effettuare la ricerca sarebbe sceso a ≈ 200.000 blocchi.




Nome	Dimensione
 unspent-0_214553	366.035 KB
 unspent-214554_446026	4.719.610 KB
 unspent-446027_501950	3.370.577 KB
 unspent-501951_610680	3.889.585 KB
 unspent-610681_698461	2.567.750 KB

Figura 3.12: File *unspent.csv*

Questa analisi rimane ipotetica perché, sebbene il codice compilasse, non è stato possibile ottenere un risultato in tempo.

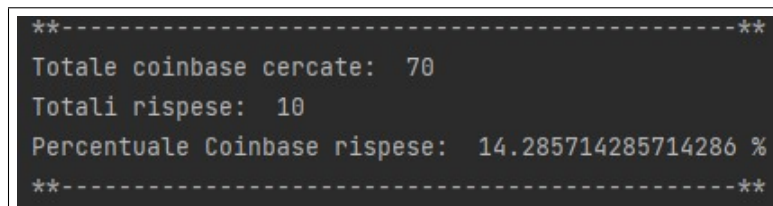
3.8 Analisi Coinbase

Come spiegato nella Sezione 1.8, ogni transazione è costituita da n Input e m Output, ad eccezione della Coinbase, che è l'unica transazione che ha solamente l'Output. Difatti è la transazione che contiene la *reward* per il miner che ha minato il blocco e validato le transazioni al suo interno. Inoltre, questa è la prima transazione del

blocco, infatti è la prima che il minatore inserisce. Si è deciso di analizzare quante di queste Coinbase vengono poi rispese dai minatori. Non potendo analizzare tutti e settecentomila blocchi minati, si è scelto di prenderne uno un campione preso in modo uniforme su tutta la Blockchain Bitcoin, a partire dal blocco 0 fino al blocco 698461.

Tramite il comando **csvdump** si è ottenuto il file *tx_out.csv*, nel formato:

`txid ; indexOut ; height ; value ; scriptPubKey ; address`. Tramite la libreria *subprocess* si è utilizzato il comando **csvdump** a tempo di esecuzione sul campione di blocchi e utilizzando una lista, o una lista di dizionari, si sono salvati gli identificativi delle Coinbase, nel campo *"txid"*. La Coinbase è la prima transazione del blocco, perciò è stata immediata da individuare. Una volta ottenuta una lista di Coinbase è stato possibile verificare che fossero state rispese ricercandole nei file *unspent.csv* già generati.



```
**-----**
Totale coinbase cercate: 70
Totali rispese: 10
Percentuale Coinbase rispese: 14.285714285714286 %
**-----**
```

Figura 3.13: Analisi Coinbase

L'analisi è stata fatta su un campione di esigue dimensioni sia per motivi di tempo computazionale necessario, sia perché il vero interesse era la buona riuscita dell'analisi. I risultati non sorprendono: buona parte delle Coinbase sono rimaste nelle tasche dei minatori senza esser rispese.

3.9 Analisi Miner

Un Miner è un nodo che ha minato almeno un blocco nella storia di Bitcoin e che ha, di conseguenza, ricevuto la sua meritata *reward*. Dato che la Coinbase è la transazione che conia nuova moneta, sotto forma di *reward* per il minatore che ha investito le proprie risorse, seguendo la "scia" di queste *reward* è possibile risalire agli *address* dei minatori. Di conseguenza gli *address* contenuti nella Coinbase appartengono a dei minatori e, utilizzando lo stesso codice Python utilizzato nell'analisi 3.8, si può risalire agli *address* di questi minatori. Prima di tutto bisogna chiarire il fatto che, per motivi di sicurezza, i *miner* non utilizzano gli stessi *address*, infatti è logico pensare che per ogni blocco minato ne generino uno nuovo.

Come nell'analisi 3.8, si può prendere un campione di Coinbase scelto in modo uniforme su tutta la Blockchain Bitcoin, tramite la libreria *subprocess* si può richiamare, a tempo di esecuzione, il comando **csvdump** e generare il file *tx_out.csv*. In questo caso il campo da salvare è "*address*". Questo codice è stato accorpato al codice per analizzare le Coinbase per risparmiare tempo e risorse.

Una volta ottenuta una lista di *address* di Miner, si può ricercare ogni singolo *address* nel file complessivo di *balance* calcolato in precedenza in (3.2.1). A scopo di esempio è stato preso l'*address* presente dal blocco genesis (1.5.2) ed è stato ricercato per verificarne il *balance*.

```
**-*****-  
L'address ricercato è: 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa  
Il suo balance è complessivo: 68.52163145  
**-*****-
```

Figura 3.14: Analisi Address Miner

In questo caso la struttura dati migliore è una lista di dizionari, perché è facile

da modificare dinamicamente. Ogni dizionario conterrà i campi "*address*" e "*balance*", quest'ultimo ha bisogno di essere aggiornato dinamicamente, una volta ritrovato l'*address* del *miner* nei file *balance.csv*.

Chiaramente non è possibile risalire ai proprietari dei vari *address*, come non è possibile sapere quanti *address* sono associati ad uno stesso minatore. Fare un vero e proprio censimento completo dei *miner* è un problema molto complesso e non è stato oggetto di questo lavoro.

Capitolo 4

Conclusioni e sviluppi futuri

La Blockchain di Bitcoin è una fonte di dati pubblica, accessibile a tutti coloro che ne sono interessati e che sono disposti a far parte del network Bitcoin. Le analisi dati che si possono fare su di essa sono di varia natura, a seconda del campo di interesse. Si possono fare analisi di natura finanziaria, come fatto in questo lavoro, ad esempio nell'analisi 3.2 e 3.3. Da questo lavoro sono state escluse analisi di altra natura, come la ricerca di messaggi nascosti all'interno della Blockchain che alcuni utenti si divertono a lasciare.

In questo lavoro sono stati utilizzati determinati strumenti per estrapolare dati, ed è stato necessario sviluppare tecniche per maneggiare l'enorme quantità di dati da analizzare per derivarne informazioni. Alcuni prototipi di programmi, come quelli sviluppati in 3.8 e in 3.9, ritengo siano un buon punto di partenza per approfondire alcuni aspetti di questo mondo.

Quello che si può notare al termine di questo lavoro è la difficoltà di calcolo di un'analisi dati del genere. È evidente come, seppur si abbiano tutti i dati necessari a disposizione, ci siano limitazioni tali, dati anche dalla potenza di calcolo necessaria, da non poter approfondire alcuni aspetti di questa nuova affascinante moneta che farà

parte dell'Internet ancora per diversi anni a venire.¹

¹A meno che qualcuno non si metta ad utilizzare computer quantistici.

Ringraziamenti

Il primo grande ringraziamento va al Professore Francesco Pasquale, per la professionalità, genialità e cura che mette nel proprio lavoro. E soprattutto grazie per la pazienza!

Grazie a Sara, per essere la sorella e amica migliore che si possa avere.

Grazie a Luca, l'Orso/criceto di cui tutti avrebbero bisogno nelle proprie vite, senza meritarselo mai.

Grazie a Nadia e Fabio, per rendermi ogni giorno la migliore versione di me.

Grazie a Gabriele, per aver fatto l'Università in simbiosi con me. Questo percorso non sarebbe stato lo stesso senza di te.

Grazie a Filippo per avermi messo il dubbio dei computer quantistici...

Grazie ai ragazzi e alle ragazze del gruppo, per essere loro stessi.

Grazie a Flavia, che rimane l'amica di sempre, anche non sentendoci tutti i giorni.

Grazie a Marco, ancora ricordo quando mi hai fatto compagnia all'esame di Fondamenti di Informatica!

Grazie ad Andrea e Francesco, per la compagnia serale.

Grazie a Fatima, per ascoltare i miei audio infiniti contenenti scleri di vario genere.

Grazie a Daniela, per il supporto dentro e fuori l'acqua.

Grazie a Chester Bennington, sempre nelle mie orecchie.

Elenco delle figure

1.1	Sistema Centralizzato e Decentralizzato	10
1.2	Esempio logico di blocchi consecutivi all'interno della blockchain . . .	12
1.3	The Times del 3 Gennaio 2009	13
1.4	Schermata del proprio wallet all'avvio di Bitcoin Core	14
1.5	Modello UTXO	20
1.6	Transazione Coinbase con <i>reward</i> , l'unica transazione ad avere solo l'Output. Sono i nuovi <i>coin</i> generati	21
1.7	Struttura di una transazione	21
2.1	Potenzialità di rusty-blockparser	24
2.2	Tabella riassuntiva dei comandi e dei <i>file</i> generabili di Rusty-blockparser	25
2.3	<i>File transactions</i> generati per le analisi svolte in questo lavoro. . . .	27
3.1	Divisione temporale	31
3.2	Bilancio degli address	32
3.3	Grafico dell'andamento del bilancio degli indirizzi	33
3.4	Analisi UTXO	34
3.5	Grafico analisi UTXO	35
3.6	Analisi contenente il balance medio e il valore medio delle UTXO . .	35
3.7	Grafico contenente il balance medio e il valore medio delle UTXO . .	36

3.8	Analisi blocksize	37
3.9	Grafico analisi blocksize	38
3.10	Analisi Transazioni	39
3.11	Pseudocodice Ricerca Binaria	41
3.12	File <i>unspent.csv</i>	42
3.13	Analisi Coinbase	43
3.14	Analisi Address Miner	44

Bibliografia

- [1] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the open blockchain*. " O'Reilly Media, Inc.", 2017.
- [2] Barbara Bachmann. *Una cascata di soldi*. Internazionale, 26 Feb/4 Mar 2021.
- [3] *Bitcoin Core*. URL: <https://bitcoin.org/en/bitcoin-paper>.
- [4] *Bitcoin Core*. URL: <https://bitcoin.org/en/bitcoin-core/>.
- [5] *Bitcoin Talk*. URL: <https://bitcointalk.org/index.php?action=profile;u=3>.
- [6] *bitcoin.org*. URL: <https://bitcoin.org/en/full-node#costs-and-warnings>.
- [7] *blockchain.com*. URL: <https://www.blockchain.com/btc/block/0>.
- [8] *forbes*. URL: <https://www.forbes.com/advisor/investing/what-is-a-bitcoin-wallet/>.
- [9] *GitHub*. URL: <https://github.com/>.
- [10] Johannes Göbel e Anthony E Krzesinski. «Increased block size and Bitcoin blockchain dynamics». In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE. 2017, pp. 1–6.
- [11] Eric Hughes. *A Cypherpunk's Manifesto*. 1993. URL: <https://nakamotoinstitute.org/static/docs/cypherpunk-manifesto.txt>.
- [12] Leslie Lamport, Robert E. Shostak e Marshall C. Pease. «The Byzantine Generals Problem». In: *ACM Trans. Program. Lang. Syst.* 4 (1982), pp. 382–401.
- [13] *Money Control*. URL: <https://www.moneycontrol.com/news/business/cryptocurrency/bitcoin-pizza-day-2021-some-interesting-facts-about-this-special-cryptocurrency-day-6924731.html>.
- [14] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009.
- [15] *pandas*. URL: <https://pandas.pydata.org/>.
- [16] *rusty-blockparser*. URL: <https://github.com/gcarq/rusty-blockparser>.
- [17] *SHattered*. URL: <https://shattered.io/>.