Blockchain Python (autore: Vittorio Albertoni)

Cos'è una blockchain

La blockchain è un modo di immagazzinare dati in formato digitale.

La prima applicazione si è avuta per i dati relativi a transazioni effettuate utilizzando una moneta virtuale, il così detto Bitcoin, ma può trattarsi di qualsiasi altro dato.

Il punto di forza di questo modo di immagazzinare dati sta nel fatto che il dato viene inserito in un contenitore, chiamato block, e i contenitori dei dati vengono incatenati tra loro attraverso la condivisione di un elemento, chiamato hash, che garantisce la persistenza e l'immutabilità dei dati inseriti nei contenitori.

Diventa così possibile realizzare un database che tutti possono alimentare, che tutti possono consultare ma che nessuno, una volta inserito il dato, può modificare.

Un database con queste caratteristiche si presta ad essere allocato in maniera distribuita in rete in modo che chiunque vi possa accedere per alimentarlo o per consultarlo, sapendo che nessuno potrà mai modificare i dati che già contiene.

Ovviamente questa è una potenzialità estrema e rimane comunque la possibilità di regolamentare attraverso i più svariati accorgimenti gli accessi per l'alimentazione o per la consultazione.

Python per la blockchain

Le tre cose fondamentali per realizzare una blockchain (blocco, hash e catena) possiamo tranquillamente costruirle con la dotazione di base di Python senza bisogno di moduli aggiuntivi.

Ci servono semplicemente la struttura di dati `lista` di Python per costruire la catena e il modulo `hashlib` per generare l'hash di collegamento dei blocchi: questo collegamento si realizza inserendo in ciascun blocco l'hash del blocco precedente.

La lista è una serie ordinata di elementi contrassegnati da un indice ed è racchiusa tra parentesi quadre. E' la struttura dati di Python che meglio si presta per rappresentare la catena di blocchi.

L'hashlib è il modulo che contiene una funzione (algoritmo di hash) che, elaborando una mole qualsiasi di bit, cioè un qualsiasi dato in forma digitale, restituisce una stringa di numeri e lettere che ha le seguenti caratteristiche:
. ha lunghezza fissa non dipendente dalla dimensione del dato di partenza,
. non consente di risalire al dato di partenza (l'algoritmo non è invertibile),
. cambia completamente se si modifica un solo bit nel dato di partenza, garantendo una corrispondenza univoca tra la stringa generata e il dato di partenza.

L'incatenamento dei blocchi avviene inserendo in ciascun blocco della catena l'hash del blocco che lo precede.

Vediamo come realizzare queste primissime cose di base con il linguaggio Python.

Ci serve innanzi tutto una classe per costruire il blocco, dotata di una funzione membro per calcolare l'hash, e la possiamo concepire così:

```
class Blocco:
   def __init__(self, indice, dato, hash_precedente):
       self.indice = indice
       self.dato = dato
       self.hash_precedente = hash_precedente
       self.hash = self.calcola_hash()
   def calcola_hash(self):
       chiave = hashlib.sha256()
       chiave.update(str(self.indice).encode('utf-8'))
       chiave.update(str(self.dato).encode('utf-8'))
       chiave.update(str(self.hash_precedente).encode('utf-8'))
       return chiave.hexdigest()
```

Il costruttore incamera i dati essenziali che ho ritenuto sufficienti per questa esemplificazione: un indice, il dato (numero o stringa) e l'hash del blocco precedente. Sempre il costruttore richiama la funzione membro per calcolare l'hash del blocco.
La funzione per calcolare l'hash si avvale dell'algoritmo sha256, lo stesso usato per i Bitcoin, e lo calcola procedendo per gradi su ciascun elemento del blocco generando un digest esadecimale (il digest è l'output dell'algoritmo).
Per utilizzare la classe occorre importare il modulo `hashlib`.

Poi ci serve una classe per incatenare i blocchi, che potrebbe essere concepita così:

```
class Catena:
   def __init__(self):
       self.catena = [self.primo_blocco()]
   def primo_blocco(self):
       return Blocco(0, '0', '0')
   def aggiungi_blocco(self, dato):
       s.catena.append(Blocco(len(self.catena), dato, self.catena[len(self.catena)-1].hash))
```

Il costruttore della catena inserisce in una lista un primo blocco, generato da una funzione membro con dati azzerati. Questo primo blocco, fittizio, serve unicamente per calcolare l'hash da inserire nel primo blocco effettivo.
Il primo blocco effettivo viene inserito con la funzione membro `aggiungi_blocco` alla quale viene passato il solo argomento `dato`, in quanto gli altri due se li trova la funzione stessa: il primo, l'indice, contando gli elementi della catena e l'ultimo andando a prendere l'hash del blocco precedente (elementi della catena -1).

Bastano queste due classi per costruire una blockchain minimale, appena sufficiente per capire di che cosa stiamo parlando.

Un piccolo esempio

Con una blockchain potremmo gestire un diario in comune tra un gruppo di amici: un diario nel quale ciascuno possa inserire sue notizie e suoi pensieri e tutti vedano e condividano.

Trattandosi di un diario è bene che nei blocchi vi sia una data.

La classe per il blocco diventa

```
import hashlib
class Blocco:
    def __init__(s, indice, data, testo, hash_precedente):
        s.indice = indice
        s.data = data
        s.testo = testo
        s.hash_precedente = hash_precedente
        s.hash = s.calcola_hash()
    def calcola_hash(s):
        chiave = hashlib.sha256()
        chiave.update(str(s.indice).encode('utf-8'))
        chiave.update(str(s.data).encode('utf-8'))
        chiave.update(str(s.testo).encode('utf-8'))
        chiave.update(str(s.hash_precedente).encode('utf-8'))
        return chiave.hexdigest()
```

E' quanto visto prima, con inserita la data e con la dizione `testo` ad indicare ciò che viene scritto nel diario. Per semplificare ho anche sostituito una semplice `s` a `self`. Infine ho utilmente indicato l'importazione della libreria hashlib.

Conseguentemente dobbiamo adattare la classe per la catena, che diventa

```
import datetime
from Blocco import Blocco
class Catena:
    def __init__(s):
        s.catena = [s.primo_blocco()]
    def primo_blocco(s):
        return Blocco(0, datetime.date.today(), '0', '0')
    def aggiungi_blocco(s, testo):
        s.catena.append(Blocco(len(s.catena),
                               datetime.date.today(),
                               testo,
                               s.catena[len(s.catena)-1].hash))
    def dimensione_catena(s):
        return len(s.catena)-1
    def contenuto_blocco(s, i):
        s.i = i
        return [str(s.catena[i].data), s.catena[i].testo]
```

Ho ritenuto utile inserire una funzione membro che dà conto dei blocchi effettivi contenuti nella catena (il -1 toglie dal conteggio il primo blocco fittizio).
Infine ho inserito una funzione membro per leggere il dato contenuto nel blocco di indice i.

Per sperimentare il tutto salviamo le due classi, rispettivamente, in due file: Blocco.py e Catena.py.

Nella stessa directory dove abbiamo salvato le due classi creiamo il file Diario.py contenente il seguente codice:

```
from Blocco import Blocco
from Catena import Catena
Diario = Catena()
Diario.aggiungi_blocco("Oggi ho finalmente capito qualche cosa della blockchain")
print(Diario.dimensione_catena())
print(Diario.contenuto_blocco(1))
```

Lanciando quest'ultimo script otteniamo questo output

```
1
['2019, 11, 2', 'Oggi ho finalmente capito qualche cosa della blockchain']
```

Con il nostro script abbiamo costruito una blockchain chiamata Diario, vi abbiamo aggiunto un blocco contenente una considerazione, abbiamo chiesto di stampare la dimensione della blockchain dopo l'inserimento del blocco e di stampare data e contenuto del blocco 1.
Ed è ciò che abbiamo ottenuto in output.

Cosa manca per fare sul serio

Lo script Diario.py del piccolo esempio è servito a verificare il funzionamento delle due classi base per costruire una blockchain ma non serve ad altro.

Il nostro obiettivo era, infatti, costruire un Diario al quale potessero accedere i nostri amici e che ciascuno della cerchia di amici potesse consultare, senza che nessuno potesse modificare il contenuto.

Quanto fatto nel precedente paragrafo risiede sul nostro computer e per gestire il nostro diario sul nostro computer non c'è bisogno della blockchain.

Il primo problema è pertanto quello di salvare, mano a mano che viene alimentata, la nostra blockchain/Diario in rete, creando un nodo per ciascuno dei nostri amici che vogliano partecipare al Diario.

Per l'interazione con la rete dobbiamo arricchire la nostra dotazione Python di due moduli che non fanno parte della dotazione di base: Flask e requests. Li possiamo caricare con pip.

L'altro problema riguarda le modalità con cui vengono memorizzati i blocchi in rete.

La garanzia della non modificabilità dei dati inseriti nella blockchain è data dal fatto che ciascun blocco contiene l'hash del blocco precedente. Pertanto, se qualcuno riuscisse a modificare i dati di un blocco, l'hash di quel blocco si modificherebbe e non corrisponderebbe più a quello prima inserito nel blocco successivo e, per mantenere funzionante la catena, occorrerebbe ricalcolare tutti gli hash dei blocchi successivi.

Tutto questo, con il semplice utilizzo della funzione di hashing che abbiamo visto prima sarebbe molto facile e, pertanto, la nostra catena non sarebbe gran che sul piano della sicurezza.

Fortunatamente la funzione di hashing si presta a fornire risultati condizionati: possiamo, per esempio, stabilire che l'hash generato abbia uno zero come primo carattere e stabilire così il grado di difficoltà 1 per generare l'hash: più zeri pretendiamo come primi caratteri più si alza la difficoltà.
Quando la funzione di hashing trova una soluzione, che non è detto sia l'unica, genera l'hash e un numero pseudo-casuale ad utilizzo unico, chiamato nonce (che significa «per l'occasione») a prova del fatto che si tratta di una soluzione valida: in gergo questa prova si chiama proof of work.
Il titolare del nodo della catena che ha trovato la soluzione è abilitato ad inserire il blocco nella catena: è quello che si chiama «miner».
Se il grado di difficoltà è elevato (pare che la pretesa di quattro zeri all'inizio dell'hash rappresenti una difficoltà molto elevata) l'impegno computazionale diventa veramente notevole per il miner e diventa praticamente nemmeno affrontabile per chi voglia ricostruire tutti gli hash relativi ai blocchi che compongono una catena semplicemente per introdurre qualche modifica. Da qui l'immutabilità della blockchain.

Approfondimenti

Dal momento che non amo riscrivere ciò che altri hanno scritto meglio di quanto possa scrivere io, a questo punto, dopo la piccola introduzione ad uso dei dilettanti neofiti cui uso rivolgermi, fornisco alcune segnalazioni a chi voglia approfondire.

Per capire e sperimentare ciò che segnalerò basta una conoscenza di base del linguaggio Python e la nostra dotazione Python (Python 3 dalla versione 6 in poi), oltre ai seguenti moduli che dovrebbero già far parte della dotazione base:

. `hashlib`, per la generazione degli hash con cui concatenare i blocchi,
. `json`, per lo scambio dei dati in formato json (JavaScript Object Notation),
. `time`, per registrare data e ora degli interventi per cui serva conservare questo dato,

deve contenere anche i moduli:

. `flask`, per creare gli endpoints del server per la blockchain,
. `requests`, per lanciare richieste http in linguaggio Python,

che non fanno parte della dotazione di base.
Per verificare se i moduli sono installati basta tentarne l'importazione in una shell Python.
Quelli per i quali viene rifiutata l'importazione vanno installati con pip o in altro altro modo[1].

Un ottimo strumento didattico per capire la blockchain ed implementarne un esemplare con il linguaggio Python lo dobbiamo a IBM: «Develop a blockchain application from scratch in Python» e lo troviamo all'indirizzo
https://www.ibm.com/developerworks/cloud/library/cl-develop-blockchain-app-in-python/index.html.
Si tratta di un tutorial che ci guida passo passo a costruire una blockchain con il linguaggio Python. Il codice finale complessivo si trova all'indirizzo
https://github.com/satwikkansal/python_blockchain_app.

[1] A chi abbia bisogno di capire consiglio il documento PDF mondo_python allegato al mio articolo "Python per tutti" pubblicato su questo blog nel febbraio 2017

Un altro ottimo strumento per capire ce lo offre Daniel van Flymen, che ci racconta, passo passo, ciò che ha fatto lui per capire, sempre utilizzando il linguaggio Python, nel documento «Learning blockchain by building one» che troviamo all'indirizzo
https://blockchain.works-hub.com/learn/Learn-Blockchains-by-Building-One.
Il codice finale complessivo si trova all'indirizzo
https://github.com/dvf/blockchain.

Meno pregevole, ma con il vantaggio di essere scritto in italiano, quanto troviamo all'indirizzo
https://www.coretech.it/it/service/knowledge_base/Programmazione/Python/Blockchain-costruiamola-in-Python.php
accessibile dopo aver creato un account gratuito su CoreTech.