Come funzionano i Bitcoin (e il Dark Web)

Uno dei punti di svolta fondamentali nel corso dell'evoluzione del genere umano è rappresentato dall'invenzione del denaro. Molti animali, infatti, conducono una vita da individui, senza mai costruire una società (pensiamo, per esempio, ai gatti). Altri animali, ma in numero inferiore, hanno imparato a vivere in società più o meno grandi: molte scimmie vivono in vere e proprie famiglie, più o meno organizzate. Ed è naturale che, quando c'è una organizzazione sociale in un gruppo, ogni individuo abbia un compito che serve alla comunità. L'individuo cede qualcosa al gruppo, ed il gruppo ridistribuisce i beni a tutti gli individui. Nasce quindi l'esigenza del baratto: una scimmia raccoglie delle banane, ed un'altra raccoglie dei mango. La prima consegnerà un paio di banane alla seconda in cambio di un frutto di mango, e viceversa. Il baratto, dunque, è una componente fondamentale della società, perché grazie ad esso non è necessario che tutti sappiano fare tutto e ci si può specializzare in un compito particolare sapendo che qualcun altro si occuperà del resto. Gli animali che hanno implementato un modello di società tendono a seguire il meccanismo del baratto. Ma solo un animale è, finora, riuscito a superare il baratto con un trucco: fissare in modo univoco il valore di un certo oggetto. È il concetto di denaro, e l'animale in questione è ovviamente l'uomo. L'acquisto di beni o servizi tramite denaro è, fondamentalmente, una forma di baratto: il cliente prende dal venditore un casco di banane in cambio di un pezzo di metallo che, per convenzione, vale 2 euro. Ma l'uso del denaro ha due vantaggi importantissimi rispetto al baratto: il primo vantaggio è che in questo modo è più facile tenere sotto controllo il valore delle cose. Infatti, un venditore può barattare un caso di banane con un ananas, mentre un altro venditore baratta lo stesso casco di banane con una capra. Ma

difficilmente si potrebbe vendere quel casco di banane per 2 euro in un negozio e 20 euro nell'altro. L'altro grande vantaggio è la possibilità di accumulare il denaro per tempi migliori: non si possono accumulare banane per aspettare di barattarle con qualcos'altro durante l'inverno o addirittura negli anni a venire, perché marcirebbero nel frattempo. Invece, nel caso del denaro, è possibile metterlo da parte per spenderlo in momenti di magra. È il concetto delle pensioni: durante la vita lavorativa si mettono da parte dei soldi, che vengono poi utilizzati per mantenersi quando non si è più in grado di lavorare. E se già così non ci piace molto l'idea di mettere da parte denaro per almeno vent'anni, mettere da parte banane e ananas per due decadi avrebbe risultati decisamente peggiori.

Table of Contents

I "bug" del denaro

L'introduzione che avete appena letto è, probabilmente, inutile: tutti noi utilizziamo denaro ogni giorno. Ma, proprio perché ne siamo tanto abituati, lo diamo per scontato non pensiamo troppo a come funzioni davvero il meccanismo dell'acquisto, in contrapposizione a quello del baratto.
Ora la domanda è: quali sono le vulnerabilità del denaro? Sicuramente, la contraffazione: fin da quando è apparso per la

prima volta, il denaro è stato soggetto a falsificazioni. I progettisti delle varie monete hanno sempre cercato, nel corso dei secoli, metodi per rendere la vita più difficile possibile ai falsari. Nel corso degli ultimi secoli si sono utilizzate monete realizzate con punzoni particolari, difficili da replicare fedelmente, mentre oggi siamo abituati a vedere sulle banconote filigrane molto particolari. Ma i falsari prima o poi riescono sempre a trovare un metodo per duplicare il denaro.
L'avvento della contabilità digitale ha in buona parte risolto questo problema, ma ne ha introdotto un altro: la tracciabilità, che può generare nella violazione della privacy.
C'è una precisazione da fare: abbiamo parlato di "contabilità digitale" e non di "denaro digitale". Infatti, quando facciamo un acquisto online tramite Visa o Mastercard, il denaro è sempre reale: la contabilità delle transazioni è digitale. In pratica, non cambia il meccanismo di pagamento: alla fin fine se acquistiamo una penna su ebay, il denaro che paghiamo arriverà in mano al venditore. Possiamo schematizzare in questo modo: noi versiamo dei contanti alla nostra banca, la somma entra nel nostro conto corrente online, paghiamo la cifra dovuta al commerciante, tale cifra viene spostata sul conto del venditore, e questo va alla sede della sua banca per prelevare, tramite bancomat, il denaro. I denaro rimane sempre sotto forma di monete o banconote, anche se le banconote che il venditore si ritroverà in mano non saranno fisicamente le stesse che noi avevamo versato in banca. Il denaro in circolo è sempre rappresentato da banconote fisiche.
Quello che cambia è il modo in cui le transazioni vengono registrate: con la contabilità "classica" è facile perdere le tracce dopo un paio di intermediari. Per esempio, se acquistiamo un ombrello pagando in banconote innanzitutto non è possibile sapere che siamo stati davvero noi ad eseguire l'acquisto e non, piuttosto, qualcun altro. In secondo luogo, il commerciante avrà segnato la cifra che gli abbiamo corrisposto tra le sue entrate, ma a sua volta è difficile

capire (anche se non impossibile) quale parte del denaro che gli abbiamo corrisposto rappresenta davvero il valore dell'ombrello e quale parte, invece, è il guadagno personale del commerciante che nulla ha a che vedere con il valore dell'oggetto in se.
Riassumendo, il problema sta nel fatto che gli acquisti sono facilmente tracciabili, e questo può rivelarsi un problema soprattutto in paesi privi delle libertà fondamentali. Un dissidente di un regime autoritario ha sicuramente il conto corrente sotto controllo.
Inoltre, il sistema è centralizzato, e i gestori possono commettere violazioni sui correntisti: l'esempio più noto è quello di Julian Assange, che dopo avere messo in imbarazzo il governo degli Stati Uniti si è ritrovato con i conti correnti congelati, impossibilitato ad eseguire qualsiasi pagamento, senza che vi fosse un provvedimento del tribunale ma per spontanea decisione degli istituti bancari.

La soluzione

Dunque, il denaro "analogico" ha due punti deboli: la falsificazione e la tracciabilità, e l'entità dei problemi varia a seconda del fatto che si usi una contabilità "tradizionale" oppure "digitale". Qualcuno, però, ha pensato ad un modo per risolvere queste vulnerabilità: passare completamente al digitale. Il programmatore Satoshi Nakamoto (è uno pseudonimo, non si conosce il nome reale di questa persona) ha proposto, nel 2008, una idea di **cryptocurrency**, cioè di "moneta virtuale cifrata". Il concetto stesso di cryptocurrency era già stato suggerito nel 1998, ma nessuno aveva mai realizzato prima una implementazione ben funzionante come quella di Satoshi Nakamoto, conosciuta col nome di Bitcoin.

I bitcoin sono legali?

Ci si potrebbe chiedere se i Bitcoin siano legali o meno. In effetti, essendo anonimi e non rintracciabili vengono utilizzati soprattutto per acquisto di materiale illecito o per ripulire denaro sporco. In realtà, però, si tratta di un bene di consumo come potrebbe esserlo una fotografia digitale. Quindi non è facile per le autorità proibirne l'uso. In Europa e negli Stati Uniti, quindi, l'uso dei Bitcoin come forma di pagamento è tollerato, anche se i governi lo sconsigliano visto che è da considerarsi sempre un investimento ad alto rischio. È capitato che le autorità statunitensi abbiano chiuso alcuni siti che consentivano di tenere portafogli Bitcoin online, requisendo le monete digitali: l'operazione è però avvenuta soltanto nei casi in cui era stata dimostrata l'esecuzione di attività illegali tramite i siti web in questione.

Come funziona una moneta digitale? Abbiamo visto che una moneta è in realtà un qualsiasi oggetto a cui viene dato un valore preciso, e che viene scambiata con altri oggetti come in una sorta di baratto. Quindi, per realizzare una moneta virtuale potremmo utilizzare un qualsiasi oggetto digitale, ed assegnargli un valore: una immagine, un file audio, una stringa di testo. Naturalmente, questi esempi non vanno bene, perché sono troppo facili da replicare: è necessario qualcosa di univoco, cioè una tipologia di oggetto in cui ogni esemplare è riconoscibile e non duplicabile (un meccanismo, quindi, simile a quello dei numeri di serie sulle banconote). La matematica ci viene in aiuto, ed ecco quindi il concetto di cryptocurrency. L'idea è di utilizzare particolari funzioni matematiche che consentono di calcolare coppie di numeri tra essi collegati ma tali da non poter risalire ad uno dei due anche se si conosce l'altro, di modo che l'unico a conoscerli entrambe sia chi li ha calcolati. È la stessa logica della

crittografia asimmetrica.

Rivest, Shamir, Adleman

Tutti noi abbiamo utilizzato la crittografia simmetrica: si sfrutta la stessa chiave per cifrare e per decifrare un messaggio. È il caso di una cifratura del tipo Cesare (cioè le lettere del testo vengono spostate, per esempio la A diventa B, la B diventa C e così via). Si tratta della forma più semplice e comune di cifratura, molti algoritmi di crittografia di uso comune sono "semplici" (per esempio quello degli archivi Zip o Rar). Questo tipo di cifratura ha un grosso svantaggio: visto che la chiave necessaria per decifrare un messaggio è la stessa usata per criptarlo, è fondamentale che il mittente invii al destinatario anche la chiave, oltre al testo cifrato. E questo complica le cose, perché se era davvero necessario proteggere il messaggio con la crittografia, non si può certo spedire la chiave senza alcuna misura di sicurezza: se qualcuno intercetta la posta, potrebbe ottenere sia il testo che la chiave, risalendo quindi al contenuto originale in chiaro (cioè non cifrato). Una soluzione al problema della distribuzione della chiave è dato dalla crittografia asimmetrica, nella quale la chiave usata per cifrare e quella necessaria a decifrare il messaggio sono diverse. Tale meccanismo era stato ipotizzato dai crittografi Diffie, Hellman, e Merkle, ed è stato reso possibile dall'algoritmo sviluppato dai matematici Rivest, Shamir, Adleman (che è per l'appunto chiamato algoritmo RSA). Come è possibile avere due chiavi differenti per cifrare e decifrare un messaggio? In realtà è piuttosto semplice, lavorando con l'aritmetica dei moduli: per capirlo meglio, vediamo come funziona l'algoritmo RSA.
Per poter utilizzare questa cifratura abbiamo innanzitutto bisogno di due numeri primi, che chiameremo **p** e **q**. In teoria, questi dovrebbero essere molto grandi (altrimenti sarebbe troppo facile riuscire a scoprirli). Ci serve, poi un altro

numero primo, più piccolo degli altri due: lo chiameremo **e**. Adesso possiamo calcolare il numero **N** come un semplice prodotto tra **p** e **q**. La chiave pubblica sarà data semplicemente dalla coppia **N** ed **e**. Quella privata, invece, si basa numeri **N** e **d**. Qui c'è un piccolo problema: il numero **d** si deve calcolare in modo tale che

*(e*d) mod ((p-1)*(q-1)) = 1*

possiamo, per semplificare, chiamare

(p-1)(q-1) = phi*

La relazione da soddisfare per ottenere **d** deve quindi essere:

*(e*d) mod (phi) = 1*

La cosa migliore per trovare **d** è andare per tentativi finchè non si identifica un numero in grado di soddisfare l'equazione. L'operazione "mod" è quella di modulo (cioè il resto della divisione). Per esempio,

7 mod 5 = 2

Poichè 7 diviso 5 fa 1 con il resto di 2. Ovviamente, **d** non dovrà mai essere rivelato perchè serve a decifrare i messaggi cifrati con e.

Come si può capire, il punto debole della crittografia RSA sta nel fatto di basare la propria sicurezza sulla difficoltà di scomporre il numero **N** in fattori primi e quindi risalire a **p** e **q**. Il problema è che la potenza dei computer aumenta ogni anno, e con essa la loro velocità nel fattorizzare un numero. Per tale motivo si cerca continuamente di costruire chiavi sempre più grandi: è una sorta di grande corsa contro il tempo in cui, per ora, i "buoni" sono avvantaggiati rispetto ai "cattivi". Ma è probabile che entro qualche decina di anni la tendenza sarà invertita, e ci resterà solo la crittografia a blocco monouso come strumento per proteggere la privacy. Ad ogni modo, per cifrare usiamo (per ogni lettera) questa relazione:

C = T^e mod N

e per decifrare quest'altra:

T = C^d mod N

Dove **C** è il testo cifrato e **T** quello in chiaro.

L’RSA è stata scoperta due volte

Rivest, Shamir, ed Adleman non sono stati i primi a scoprire la crittografia asimmetrica. Ma sono stati i primi a poterlo rendere noto. La crittografia tipo RSA venne infatti ideata da James Ellis, Clifford Cocks, e Malcolm Williamson nel 1974-1975. questi erano, però, dipendenti del centro segreto di crittografia GCHQ (servizi segreti del Regno Unito). Nonostante la loro scoperta non fosse stata immediatamente utilizzata dall’intelligence britannica, anche perchè i calcolatori non erano ancora in grado di eseguire conti così complessi, essi furono obbligati a mantenere il silenzio sulla crittografia asimmetrica, ed assistere quindi impotenti alla riscoperta dell’algoritmo da parte dei due trii di crittografi e matematici che abbiamo citato nell’articolo. Ovviamente, ciò non toglie merito ai ricercatori che oggi ricordiamo come scopritori “ufficiali” di questa tecnologia, perchè essi hanno eseguito le loro ricerche in totale autonomia, partendo da zero. Dovremmo, però, ricordare anche i dipendenti del GCHQ, che non hanno mai potuto ricevere i dovuti onori.

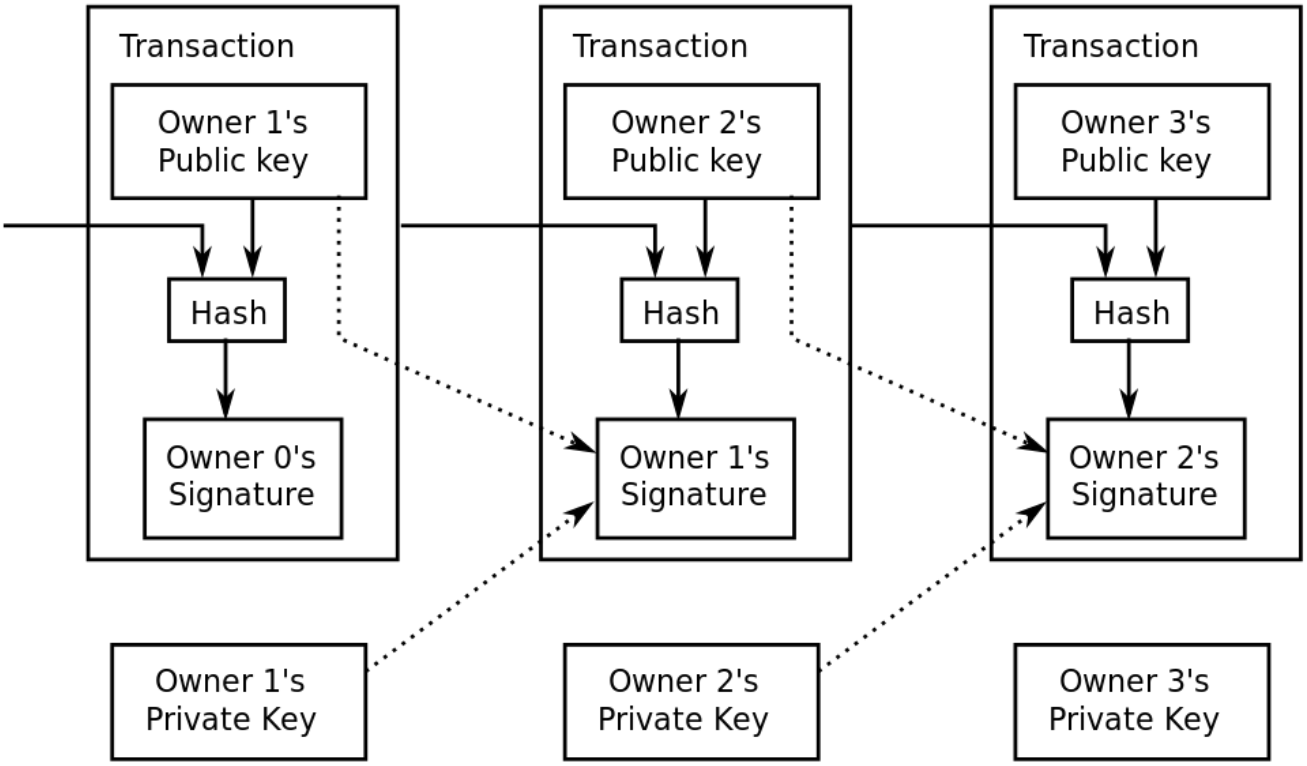
L’RSA è la più importante forma di cifratura attualmente esistente. Fondamentalmente tutta la sicurezza informatica si basa su questo algoritmo (per esempio la cifratura SSL delle pagine web, la firma digitale, ecc…). In particolare, è importante capire come funziona il meccanismo di firma digitale. Nel caso della crittografia, si rende pubblica la chiave per cifrare, mentre rimane privata la chiave per decifrare. In questo modo tutti possono scrivere un messaggio criptato ad una persona, ma solo questa persona potrà leggerlo (nemmeno il mittente originale potrà più decifrare il messaggio perché dispone solo della chiave pubblica). Invece, nel caso della firma digitale, si fa esattamente l’opposto. In pratica, la chiave utilizzata per cifrare è privata, l’altra è pubblica. Questo fa sì che un utente possa prendere un documento e cifrarlo con la propria chiave privata: l’operazione garantisce che a scrivere il documento sia stato

proprio questo utente, perché è l'unico a conoscere la chiave di cifratura. Chiunque, invece, può leggere il documento utilizzando la chiave pubblica per decifrarlo: il meccanismo garantisce anche che nessuno abbia modificato il testo dopo l'apposizione della firma perchè, se il testo viene modificato, l'algoritmo di decifrazione restituirà un testo errato.

Dai messaggi alle monete

Ora che abbiamo capito come funziona la crittografia asimmetrica per i messaggi, sarà facile comprendere il meccanismo su cui si basa Bitcoin. La rete Bitcoin è costruita come una rete P2P (come Kadmilla, per esempio). Ogni utente della rete ha diverse (potenzialmente illimitate) coppie di chiavi, che può raccogliere in un portafoglio digitale. In ciascuna di queste coppie, la chiave pubblica è resa nota a tutti e funziona da punto di invio o di ricezione per i vari pagamenti: è in un certo senso un po' come un IBAN. Per tale motivo la chiave pubblica, che per essere precisi è costituita da 33 caratteri (scelti tra lettere e numeri), viene chiamata indirizzo bitcoin. Invece, la chiave privata corrispondente è nota esclusivamente al suo proprietario e serve a consentire il pagamento (cioè la cessione, non la riscossione). In questo modo, solo il proprietario della coppia di chiavi può davvero autorizzare la cessione di alcune sue monete Bitcoin. Un Bitcoin viene "creato" come ricompensa per il calcolo di un blocco (vedremo tra poco che significa) ed è una stringa di testo, contenente l'indirizzo del proprietario, cifrata con la chiave privata del proprietario stesso.
In seguito, se il proprietario vorrà cedere la sua moneta a qualcun altro per eseguire un acquisto, dovrà aggiungere ad essa l'indirizzo (la chiave pubblica) del nuovo proprietario e firmare il tutto con la propria chiave privata. Il fatto che si debba cifrare con la propria chiave privata fa sì che solo l'attuale proprietario possa porre la firma necessaria a concludere l'acquisto. Ciò significa che è sempre possibile

risalire alla storia di una moneta: tramite l'ultima chiave pubblica è possibile decifrare la parte criptata. Questa è costituita da due parti: un'altra chiave pubblica e un altro "blob" criptato (da decifrare con l'indirizzo bitcoin appena citato). A sua volta, questo blob contiene un ulteriore indirizzo bitcoin e relativo blocco di testo criptato con esso. È quindi possibile proseguire fino alla stringa di testo originale, ricostruendo l'intera storia della moneta dal suo ultimo acquirente fino al creatore ottenendo tutti i vari indirizzi bitcoin attraverso cui la moneta in questione è passata.



In pratica, ogni moneta Bitcoin esiste in quanto oggetto di una transazione economica: è addirittura costruita dall'insieme dei vari indirizzi che l'hanno posseduta. In fondo, ciò che conta di una moneta è poter eseguire delle transizioni: sono proprio queste il punto focale del meccanismo, quindi incentrare su queste il protocollo Bitcoin è la soluzione più semplice. Le transizioni sono pubbliche, anzi: quando si esegue un pagamento, le informazioni sulle monete (quantità di Bitcoin spesi, indirizzo del mittente e

del destinatario) vengono inviate all'intera rete. Inoltre ogni client Bitcoin, per poter funzionare, deve scaricare dalla rete gli aggiornamenti in tempo reale sulle transizioni eseguite nel mondo. In questo modo è impossibile che un utente spenda due volte una propria moneta: quando esegue la prima transazione, tutti sanno che quel particolare Bitcoin è passato ad un altro utente e dunque non è più di sua proprietà.

Ma è davvero sicuro?

Si potrebbe obiettare che se davvero è possibile conoscere tutti gli indirizzi proprietari di una moneta, e tutti conoscono tutte le monete esistenti (grazie al database di informazioni), il sistema sia tracciabile. In realtà, però, gli indirizzi bitcoin vengono generati in modo casuale, e non possono essere collegati direttamente ad una persona: è infatti consigliabile utilizzare un indirizzo diverso per ogni transazione. Facciamo un esempio: se vogliamo permettere ai lettori del nostro blog di inviarci un pagamento tramite bitcoin, dovremo fornire pubblicamente un indirizzo a cui far arrivare i pagamenti. Ciò significa che tutti, Guardia di Finanza e Polizia Postale incluse saranno in grado di conoscere questo indirizzo ed i suoi movimenti, compresi e soprattutto quelli futuri: potrebbero facilmente scoprire quali oggetti servizi acquisteremo con quelle monete. La soluzione consiste nell'avere a disposizione diversi indirizzi bitcoin ma non dirlo a nessuno. Dal momento che le chiavi pubbliche non si possono collegare alle persone in modo automatico, ma solamente se il proprietario decidere di farlo sapere a tutti, è sufficiente avere un indirizzo "pubblico", a cui tutti possono inviare denaro, e diversi indirizzi "privati"su cui trasferire le monete che arrivano a quello pubblico. In questo modo, eventuali inquirenti potrebbero sapere quante monete ci sono state cedute, ed anche quante ne abbiamo inviate ad altri indirizzi Bitcoin. Ma non potrebbero

in alcun modo sapere chi sia il reale proprietario di questi indirizzi di arrivo: certo, potrebbero sospettare che si tratti di indirizzi che appartengono comunque a noi e che sia tutta una messa in scena per far perdere le tracce, ma non potrebbero dimostrarlo. Va anche detto che in realtà la "cronologia" di una moneta si può ricostruire solo grazie al database delle transazione, e non direttamente dalla moneta stessa. Questo perchè, di norma, invece di cifrare l'intero testo che costituisce la moneta per eseguire la firma che garantisce una transazione economica, si cifra soltanto un hash (con algoritmo SHA-2) del testo in questione. Un hash è un'altra stringa di testo ottenuta con una funzione di hash dalla stringa che rappresenta la moneta. La funzione di hash è per definizione non iniettiva, quindi non è biettiva e non si può invertire. Di conseguenza, dalla moneta si ottiene l'hash, ma dall'hash non si può ottenere la moneta. Tuttavia, ogni moneta realizza un unico hash, e questo può essere realizzato da una solo moneta: due stringhe di testo, cioè due monete bitcoin, differenti non possono generare lo stesso hash. Ciò significa che chi possiede una moneta può risalire fino all'hash generato dal suo precedente proprietario, ma lì deve fermarsi: non può ricostruire il contenuto della moneta prima che questa arrivasse nelle mani della persona da cui gli stesso l'ha ottenuta. Certo, può comunque risalire all'intera cronologia senza nemmeno bisogno di avere la moneta "in mano": gli basta controllare il database comune delle varie transazioni.

Come si producono

L'ultima cosa che dobbiamo ancora scoprire sui Bitcoin è in che modo si possono ottenere. In fondo, per ogni moneta è fondamentale stabilire in che modo possa essere prodotta, per indicare anche un limite di produzione (anche nella realtà, la banca centrale europea non può stampare tutti gli euro che vuole, ci sono delle regole da rispettare).

Di solito nessuno si preoccupa di come vengano creati i bitcoin, perché esistono già strumenti in grado di farlo. Il fatto è che per degli sviluppatori è fondamentale saperlo, soprattutto perché man mano che passa il tempo servono sistemi sempre più efficienti per produrre nuovi blocchi, e quindi è necessario che qualcuno sappia come programmare questi "generatori di blocchi".
Cos'è un blocco? Abbiamo detto che tutte le transizioni vengono comunicate, sottoforma di messaggio, all'intera rete Bitcoin. E la rete memorizza questi messaggi in diversi "blocchi", che sono quindi semplicemente dei contenitori realizzati periodicamente.
Questo permette a nuovi client di scaricare facilmente l'intero elenco delle transizioni mai eseguite: basta ottenere tutti i blocchi, che tra l'altro sono identificabili (ci sono i blocchi della settimana scorsa, quelli di due mesi fa, quelli di tre anni cinque mesi e tre giorni fa, eccetera).
I Bitcoin vengono forniti agli utenti come ricompensa per avere risolto un problema matematico, operazione chiamata "mining" (letteralmente "scavare in miniera"). Il problema in questione consiste nell'identificare un numero (chiamato **nonce**) tale che dopo essere stato sottoposto due volte all'algoritmo di hash SHA-2 si ottenga una stringa che inizia con un certo numero di zeri. Naturalmente, visto che la funzione hash non è invertibile, si deve procedere per tentativi (un po' come nel caso del brute force). Il numero di zeri che devono essere presenti all'inizio della stringa viene variato automaticamente per rendere l'operazione più semplice o più complessa in modo che venga sempre generato, in media, un nuovo blocco ogni 10 minuti. La difficoltà di trovare il numero **nonce** aumenta esponenzialmente con la quantità di zeri che devono essere presenti all'inizio della stringa ottenuta con il doppio hash.
Ogni nodo della rete, rappresentato da un utente con il proprio client, si occupa quindi di calcolare questo numero ed utilizzarlo per ottenere assieme al contenuto del blocco su cui sta lavorando un hash. Quando riesce ad ottenere l'hash

che inizia con il giusto numero di zeri, comunica l'avvenuta scoperta a tutta le rete. Quando gli altri nodi della rete riconoscono che la soluzione proposta è corretta, "archiviano" il blocco, che entra dunque a far parte del passato, e cominciano ad inserire le prossime transizioni che riceveranno in un blocco completamente nuovo. L'insieme in ordine cronologico dei vari blocchi è chiamato "catena dei blocchi". In pratica, la rete Bitcoin si comporta in questo modo:

- Quando due utenti eseguono una transazione, questa viene comunicata a tutti i nodi
- Ogni nodo minatore raccoglie le nuove transizioni di cui viene informato in un blocco
- Ogni minatore calcola la funzione per risolvere il problema matematico (questo è il mining vero e proprio)
- Quando un nodo trova la soluzione, cioè il numero **nonce**, la invia a tutto il resto della rete
- I nodi accettano il blocco soltanto se le transizioni in esso contenute sono valide
- Il minatore ottiene una ricompensa in bitcoin per il blocco contenente la soluzione che ha appena scoperto
- I nodi dichiarano di avere accettato il blocco ricevuto e cominciano a lavorare su un altro blocco utilizzando l'hash del blocco appena accettato.
- Si ricomincia da capo

Per ogni blocco vengono forniti dei Bitcoin, in numero decrescente nel tempo. Infatti, nel 2008 per ogni blocco venivano forniti 50 Bitcoin, mentre dal 2012 ne vengono prodotti 25 (e così sarà fino al 2016). La ricompensa verrà dimezzata ogni 4 anni fino ad arrivare a zero: secondo questo meccanismo, sarà possibile generare al massimo 21 milioni di monete. Visto che la complessità di calcolo aumenta sempre più, è oggi praticamente impossibile ottenere qualcosa con il proprio computer: è necessario utilizzare dei minicomputer dedicati (per esempio **ASIC**) oppure collaborare con altri

minatori (per esempio tramite il sito **www.bitminter.com**). Nel caso della collaborazione, il sito si occupa di distribuire i bitcoin tra i vari utenti che hanno contribuito a risolvere il problema sul blocco. Ovviamente, visto che vengono forniti solo 25 Bitcoin e probabilmente ci sono ben più di 25 collaboratori tra cui distribuirli, ogni utente otterrà una frazione di moneta. In pratica, invece di ottenere 1 Bitcoin, riceverà 0,00X Bitcoin. Questo non è un problema, sia perché i Bitcoin possono essere divisi fino all'ottava cifra decimale, sia perché il valore dei Bitcoin può aumentare man mano che la gente li richiede.

Il valore dei Bitcoin

Il punto dolente dei Bitcoin sta proprio nel valore della moneta, ed è legato alla loro logica decentralizzata. Il motivo per cui i Bitcoin sono da considerarsi un investimento ad alto rischio è l'incredibile fluttazione che il suo valore ha avuto e continua ad avere. Un bitcoin può passare dal valore di 50 euro a quello di diverse centinaia di euro nel giro di pochi mesi. Un andamento semplicemente impossibile per le monete tradizionali. Il fatto è che, non esistendo un organismo centrale, non esiste nessuno che posso stabilire uno standard per il valore della moneta e che possa quindi prendere le dovute misure contro l'inflazione e la deflazione. Il valore dei Bitcoin è legato quasi esclusivamente al rapporto di domanda ed offerta: sono dunque gli utenti, con le loro decisioni di comprare o vendere le proprie monete che fanno salire o scendere il prezzo di un Bitcoin. E, considerando che ogni utente ragiona con la propria testa e per motivi personali, la variazione del valore della moneta è praticamente casuale.

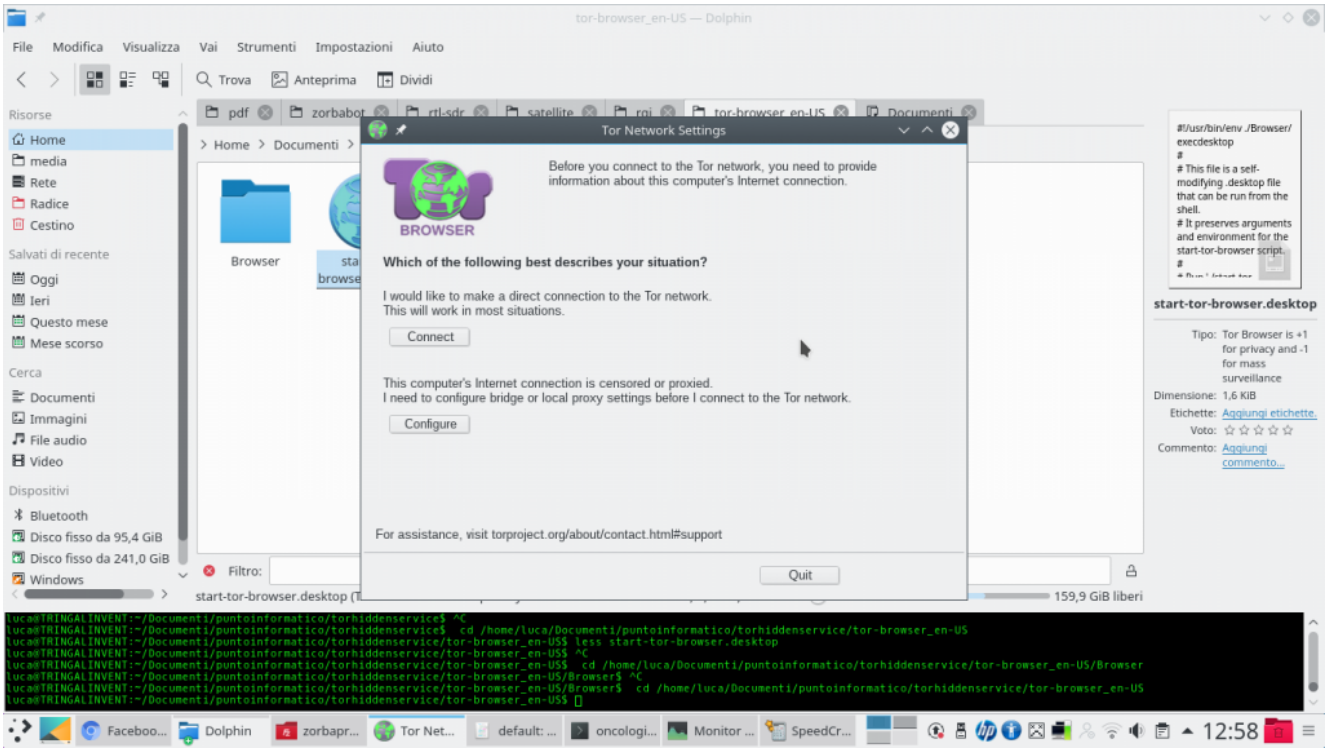
Quando i bitcoin non potranno più essere prodotti (perchè la ricompensa per ogni blocco sarà diventata pari a zero) i nodi

potranno comunque continuare a realizzare i blocchi finanziandosi in altri modi, per esempio guadagnando dalle varie transazioni gestite.

Vendere online in Bitcoin

Per accettare pagamenti in Bitcoin sul proprio sito, è possibile sfruttare diversi servizi, come accept-bitcoin, che di fatto sono il PayPal per Bitcoin. Basta aggiungere al proprio sito un codice di questo tipo:

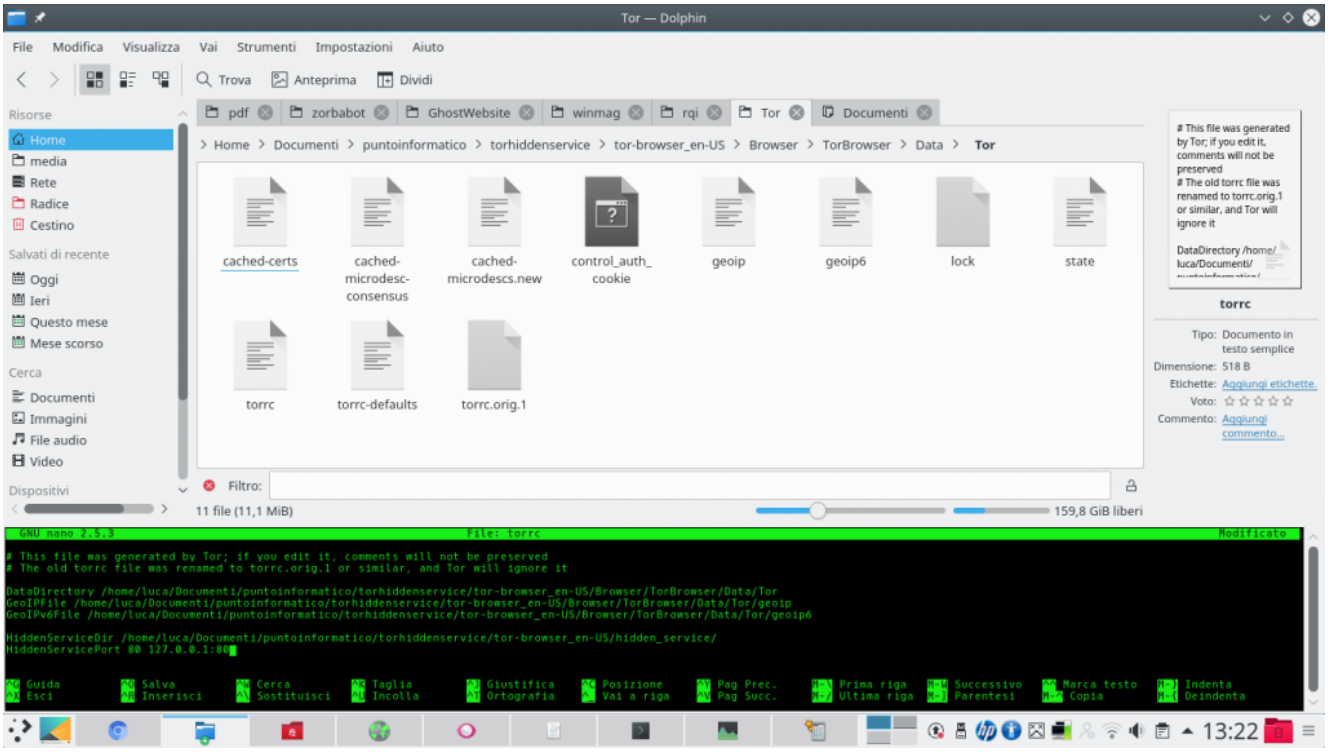
Nell'esempio stiamo vendendo 1Kg di Nduja per l'equivalente in Bitcoin di 20 euro. C'è anche una procedura guidata per realizzare questo codice, la si trova sul sito https://accept-bitcoin.net/bitcoin-payment-button.php. Ovviamente, pubblicare sul proprio sito web del materiale da pagare con Bitcoin ha poco senso: si viene facilmente tracciati. Molto meglio realizzare il proprio sito nel Dark Web, con Tor. Il bello di avere un server web che funziona sulla rete Tor, cioè un cosiddetto sito web Onion, è che non siamo identificabili: gli utenti del nostro sito web non possono risalire al nostro indirizzo IP reale, e quindi non ci possono identificare. Allo stesso modo, nessuno (noi compresi) può identificare gli utenti del nostro sito: tutti sono assolutamente anonimi.



Quando si scarica Tor Browser, la prima finestra che appare permette di configurare la connessione ad internet: nel caso ci si trovi dietro ad un proxy (come nelle reti aziendali), può essere necessario cliccare su Configure. Altrimenti, basta cliccare su Connect per iniziare la connessione a Tor. Il servizio Tor rimarrà attivo finché Tor Browser è aperto: se lo si chiude, anche la connessione a Tor viene terminata.

I siti web con dominio .onion sono per l'appunto anonimi e costituiscono il dark web, che è un sottoelemento del deep web. Il deep web più in generale rappresenta tutti quei siti non rintracciabili dai normali motori di ricerca, ma all'interno del deep web il dark web è statisticamente la parte più ampia e interessante (esistono comunque anche siti privati, che non sono accessibili da chiunque pur non essendo anonimi) e per questo i due termini sono spesso usati come sinonimi. Questo anonimato offerto dai server .onion può non essere particolarmente importante per un normale cittadino italiano, ma è fondamentale per chi si trova in paesi che limitano la libertà di stampa: un cittadino ucraino poteva (durante la rivoluzione) pubblicare un sito Tor con notizie sui crimini del governo, senza il rischio di essere scoperto ed arrestato. Per fare un altro esempio, la procedura di invio

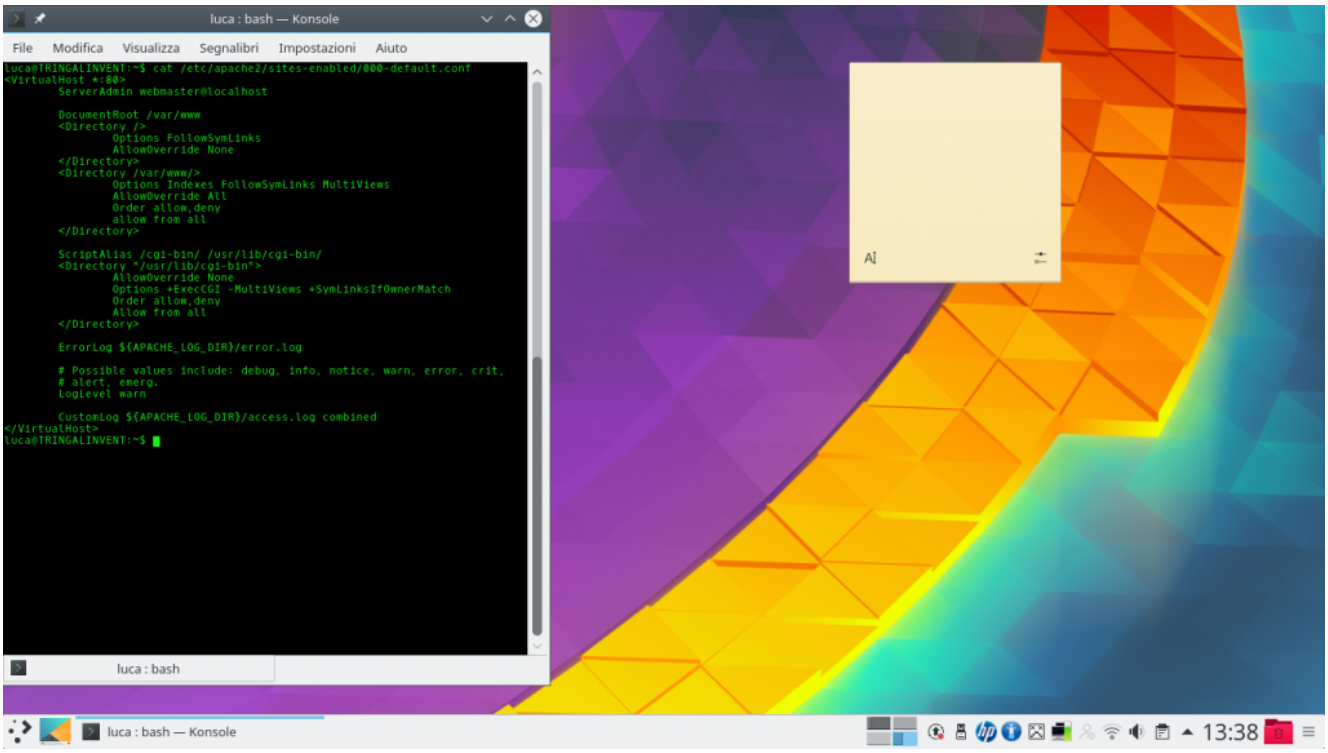
dei file da parte degli informatori a Wikileaks funzionava proprio usando un server web Tor.



Per abilitare il proprio Hidden Service è necessario aprire il file che si trova nella cartella **Browser/TorBrowser/Data/Tor/torrc** con un editor di testo, e inserire alla fine di esso le righe https://pastebin.com/EEa4PegH. In questo modo si indica come server quello che viene ospitato sulla porta 80, e come cartella dei dati /home/luca/tor-browser_en-US/hidden_service/. Ovviamente, si può scegliere una cartella qualsiasi, e una qualsiasi porta (FTP funzionerebbe sulla porta 21).

Considerata la struttura della rete Tor stessa il client Tor può essere utilizzato sia per accedere a dei servizi che per proporre i propri servizi al resto della rete. Infatti Tor è un protocollo di Onion routing, un tipo di rete simile a una VPN, con una differenza importante: ci sono molti passaggi intermedi. Una VPN è di fatto una sorta di rete locale molto estesa sul territorio, tutti i computer si collegano a uno stesso server centrale che funziona come un router domestico e

tiene assieme tutti i vari computer permettendo loro di condividere informazioni e un unico indirizzo IP pubblico. L'indirizzo IP pubblico è quello del server, quindi ogni client appare sul web con il suo indirizzo. Nell'Onion routing esistono molti passaggi intermedi: in pratica, un client si collega a un server, il quale si collega a un altro, poi un altro, e così via fino a un server finale che funziona come "punto di uscita" (un exit node). Il client apparirà sul web con l'indirizzo IP del punto di uscita, ma nessuno dei vari punti intermedi può risalire a chi sia il client e quale il nodo di uscita: ogni nodo intermedio della rete sa soltanto chi c'è prima di lui e chi dopo, ma non sa in che punto esatto della rete si trovi, quindi non sa se sta dialogando con il primo o l'ultimo computer della sequenza.



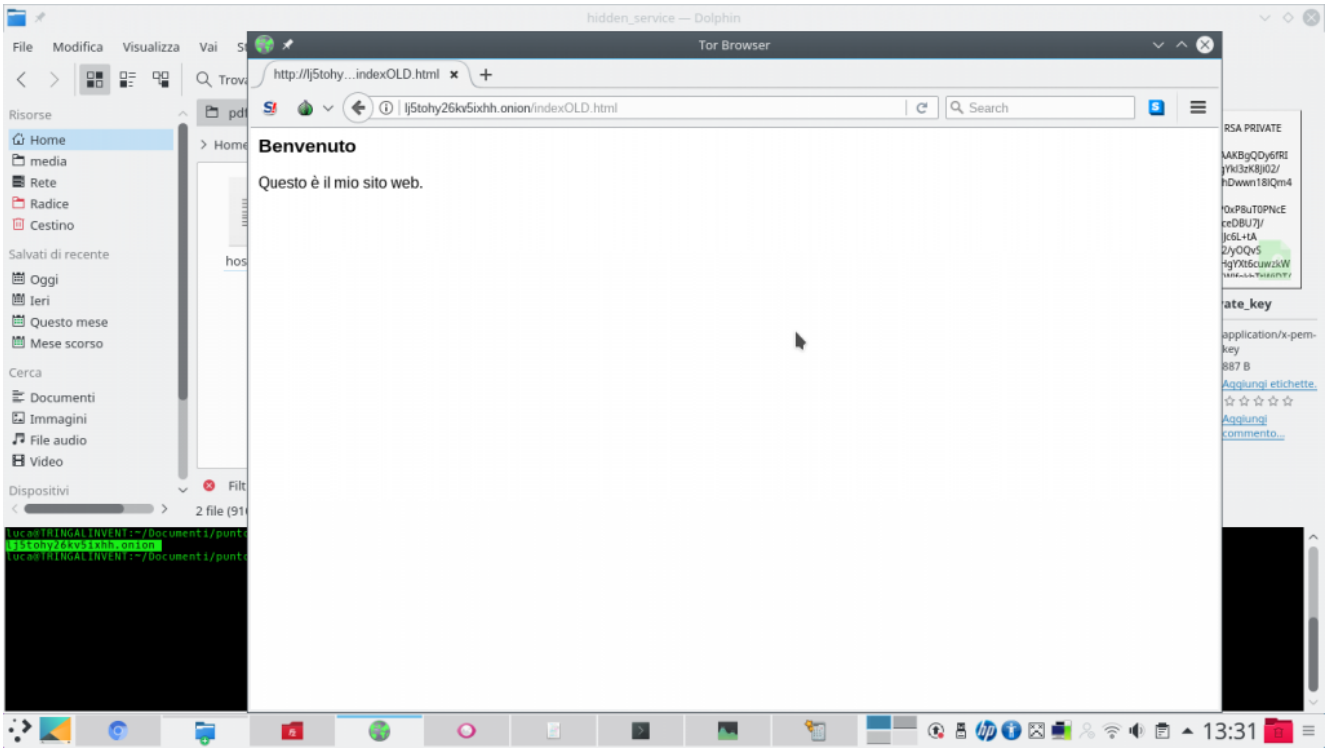
Ora è ovviamente necessario installare il server: se si vuole un server web, l'opzione probabilmente più scontata è Apache2, che può essere installato su sistemi Debian con il comando: *sudo apt-get install apache2 php5 libapache2-mod-php5 php5-mcrypt* Dando il comando *cat /etc/apache2/sites-enabled/000-default.conf* la riga DocumentRoot indica il percorso nel quale si possono inserire i vari file, per esempio la cartella

/var/www/html/. Il file index.html dovrebbe già esistere.

L’idea ricorda un po’ le reti mesh, con la differenza che nelle reti mesh solitamente è possibile per tutti i computer essere punti di uscita per qualcun altro e non si può scegliere il proprio punto di uscita. Invece, nell’Onion routing i punti di uscita sono solo alcuni dei computer coinvolti, e un client può cambiare exit node se desidera apparire con un indirizzo IP differente da quello che ha avuto finora.
Inoltre, nell’Onion routing c’è un apposito sistema crittografico: un sistema a cipolla (il nome non è casuale, del resto). Ogni router intermedio si presenta al client con una chiave crittografica pubblica, che può essere usata per crittografare i pacchetti. Immaginiamo che un client riceva le chiavi di 4 diversi router: il client preparerà un pacchetto crittografandolo con la chiave 4, la chiave 3, la 2, e infine la numero 1. Il pacchetto verrà poi inviato ai vari router in sequenza, che lo potranno decifrare man mano: il primo router toglierà soltanto la crittografia con la propria chiave privata 1, il secondo con la chiave privata 2, eccetera. In questo modo, gli unici a poter vedere i dati non crittografati sono il client e l’exit node (e se si utilizza un protocollo come HTTPS nemmeno l’exit code può vedere i dati completamente decifrati, potrà farlo soltanto il vero destinatario). Le varie crittografie sono infatti applicate a strati come in una cipolla, e devono essere tolte esattamente in quell’ordine. Se qualcuno provasse a intromettersi il procedimento salterebbe e i dati diventerebbero automaticamente illeggibili, rendendo quindi tutta la comunicazione privata e di fatto anonima perché nessuno sa chi sia davvero ad avere cominciato la connessione.

Come si accede al server Onion

Per accedere ad un server Onion non si utilizza un indirizzo IP, ma un nome di dominio che viene creato in modo casuale sulla base di una altrettanto casuale e univoca chiave crittografica. Si tratta della chiave crittografica pubblica che permette ad altri client di crittografare, come abbiamo spiegato, i pacchetti di dati in modo che soltanto il nostro server sia in grado di leggerli usando la chiave privata di decifratura. Tutto quello che si deve fare per rendere il proprio client accessibile dalla rete Tor è selezionare le porte da "aprire" al resto del dark web (nel tutorial suggeriamo come fare per la porta 80).



Per conoscere l'indirizzo del sito web sulla rete Onion, accessibile anonimamente tramite rete Tor, basta leggere il contenuto del file **tor-browser_en-US/hidden_service/hostname**. È l'indirizzo che si può condividere, anche sui motori di ricerca. Inserendo l'indirizzo su un Tor Browser è possibile visualizzare il sito appena creato. Per assicurarsi che il

sito sia accessibile solo tramite rete Tor, e non sul web, è una buona idea chiudere la porta 80 sul proprio router.

C’è però una differenza importante con i server che si abilitano sul proprio computer per l’accesso dal web convenzionale: non si passa attraverso il meccanismo del NAT, perché il client Tor è già connesso direttamente alla rete Onion. Le normali reti locali casalinghe sono infatti gestite da un router, e sono quindi “nattate”: questo significa, in parole povere, che l’unico computer visibile direttamente da internet è il router stesso. Gli altri dispositivi della rete non sono normalmente visibili, e le loro porte di comunicazione (il server web utilizza la numero 80) sono chiuse. È quindi fondamentale, se vogliamo che il nostro server sia raggiungibile da internet, aprire la porta 80 del nostro computer. Questa procedura è detta “port forwarding”: se controlliamo il manuale del nostro router troveremo sicuramente una pagina in cui viene spiegato come eseguirla. Tuttavia, su rete Tor il server funziona anche se non abbiamo abilitato il port forwarding, perchè la connessione viene gestita direttamente dalla rete Onion. Anzi: è molto meglio non aprire la porta 80 sul router. Così, l’unico modo per accedere al server sarà tramite la rete Tor: un utente (e questo vale anche per i programmi di scansione automatica come Echelon o Prism) che si trova su internet (e che quindi può leggere il nostro vero indirizzo ip) non sarebbe in grado di entrare nel sito. Se aprissimo la porta sul nostro router casalingo, il nostro server diventerebbe accessibile dal web normale, non anonimo, e prima o poi qualcuno lo troverebbe (potrebbe finire sul motore di ricerca Shodan). Se la teniamo chiusa, l’unico modo per arrivare al nostro server web sarebbe proprio attraverso il client Tor che abbiamo attivo sul computer, quindi saremmo protetti dal meccanismo di anonimato.

Le ultime versioni di Tor Browser contengono Selfrando, sistema sviluppato dall'università di Padova per proteggere il programma dall'esecuzione remota di codice

---

Facebook Scraping: scaricare tutti i post delle pagine Facebook

Da quando sono esplosi gli scandali relativi all'uso dei dati dei social network, come quello di Cambridge Analytica, Facebook ha messo in piedi un sistema di controllo delle applicazioni. In poche parole, ora qualsiasi applicazione voglia accedere a ogni tipo di dato degli utenti deve prima superare un controllo in cui, in teoria, Facebook dovrebbe verificare che l'app non usi i dati in modo contrario alle norme di condotta previste dal social network.
Il problema è che, al momento, il sistema non funziona bene: ovviamente è appena partito, e si presume che in futuro verrà

migliorato, ma ha una serie di difetti fondamentali che saranno difficili da correggere a meno che Facebook non sia pronto a spendere davvero molte risorse finanziarie. Già adesso, infatti, ci sono delle persone incaricate di analizzare ogni app che viene sottoposta alla verifica, ma hanno migliaia di app da seguire e non hanno quindi il tempo di entrare nei dettagli. Soprattutto, nessuno controlla il codice sorgente delle app, quindi non c'è davvero una garanzia che questo controllo serva a impedire utilizzi impropri dei dati del social network. Allo stesso tempo, inoltre, i meccanismi di autorizzazione delle app offerti al momento sono insufficienti a coprire alcune delle app più legittime: parliamo di quelle che collezionano dati pubblici per realizzare statistiche (per pubblica amministrazione, università, e ricerca). Al momento è molto difficile farsi approvare una app di tipo desktop, l'opzione non è prevista e gli script non sono visti di buon occhio. Tuttavia, una università, un istituto di statistica, o una redazione giornalistica hanno in genere bisogno di accedere soltanto a dati come i vari post delle pagine dei personaggi famosi (per analizzare il loro linguaggio e capire come cambi la comunicazione in caso di eventi importanti, o controllare la veridicità delle affermazioni). Un esempio semplice è un team di ricercatori che voglia controllare sistematicamente la percentuale di verità dei post di un politico, il cosiddetto fact checking. In questi casi si vuole accedere soltanto a dati che sono già pubblici, e che quindi possono essere raccolti senza violare la privacy di nessuno.

Uno script con Python

Per il nostro script utilizziamo Python3: nonostante sul sito ufficiale venga ancora presentato il vecchio Python2 per questioni di retrocompatibilità, la versione 3 presenta delle

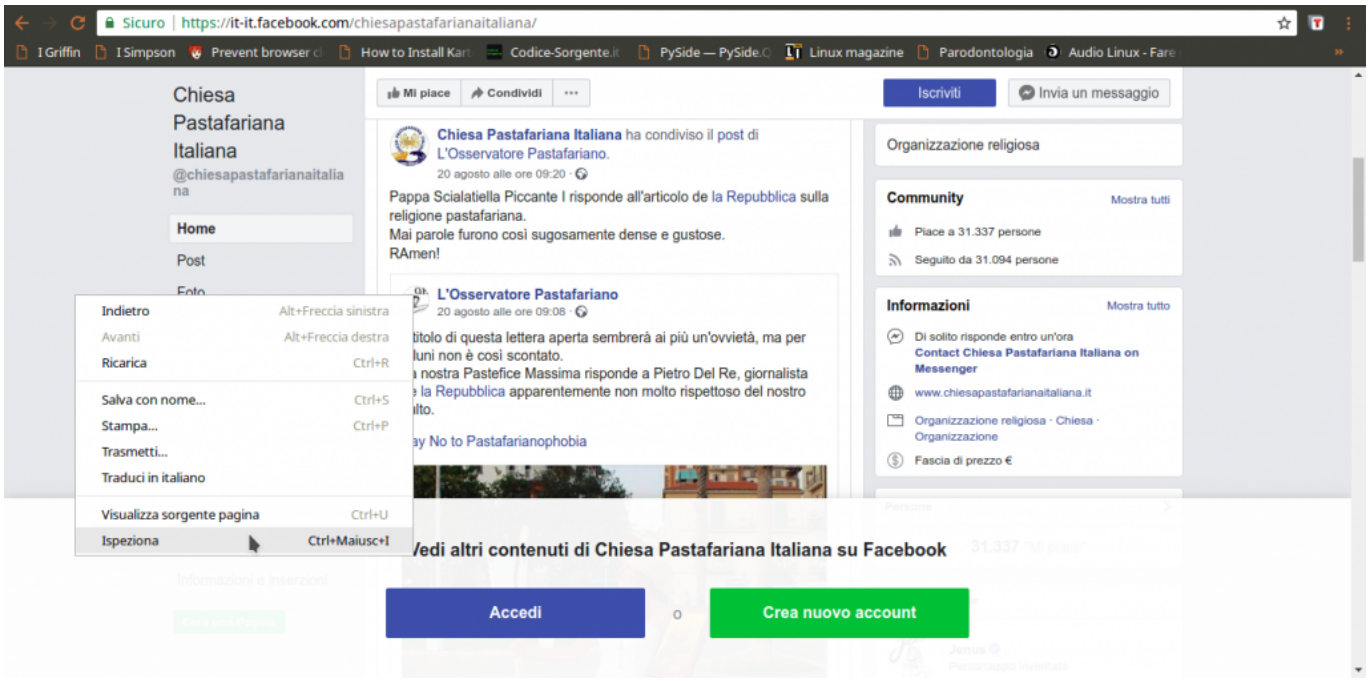
differenze importanti che rendono alcune funzioni incompatibili. Per essere sicuri di poter utilizzare lo script che presentiamo (alla fine dell'articolo trovare un link all'intero codice sorgente), bisogna installare sul proprio pc almeno la versione 3.6 di Python.

Abbiamo quindi pensato di realizzare uno script in Python che si occupi di eseguire lo scraping delle pagine Facebook pubbliche. Lo scraping è, per chi non lo sapesse, un insieme di tecniche di estrazione di informazioni da pagine web e altri documenti, in modo automatico, ripulendole da ciò che non serve. Un ricercatore universitario potrebbe scaricarsi i post di una pagina Facebook aprendola col browser e scorrendola verso il basso fino a visualizzarli tutti, selezionando il testo di ciascuno e copiandoselo. Ma sarebbe una operazione lunghissima e noiosa. Analizzando il codice delle pagine HTML che Facebook fornisce, invece, possiamo automatizzare l'estrazione dei testi (o delle immagini, se volete scaricarvi i meme delle vostre pagine preferite, basta modificare lo script per cercare i tag **img** invece dei tag **p**). E ovviamente lo script che realizziamo non richiede alcun accesso alle API o approvazione da parte di Facebook, perché di fatto faremo la stessa cosa che fa ogni utente che vuole guardare una pagina Facebook, permettendoci di bypassare tutte le verifiche che Facebook ha messo in piedi per le app.
Non dobbiamo, infatti, dimenticare un concetto fondamentale: se una informazione è disponibile, c'è sempre un modo non ufficiale per ottenerla. Quando carichiamo una pagina Facebook in Google Chrome, l'interfaccia realizzata con HTML e Javascript carica solo un certo numero di post. Quando "scrolliamo" la pagina, scendendo verso il basso, deve esserci una qualche funzione che si accorge che stiamo scendendo e quindi richiede al server un certo numero di nuovi post da visualizzare. È abbastanza ovvio che questa richiesta debba essere fatta, dalla pagina HTML+JS, con una richiesta HTTP (usando il meccanismo Ajax, quindi la funzione xmlhttprequest

di Javascript). Se ne deduce che da qualche parte all'interno della pagina ci deve essere un riferimento a un'altra pagina che fornisce un elenco di post da visualizzare. L'accesso a questi dati da parte di script automatici invece che dai normali browser web è una cosa che, a prescindere dai suoi sforzi, Facebook non potrà mai impedire.

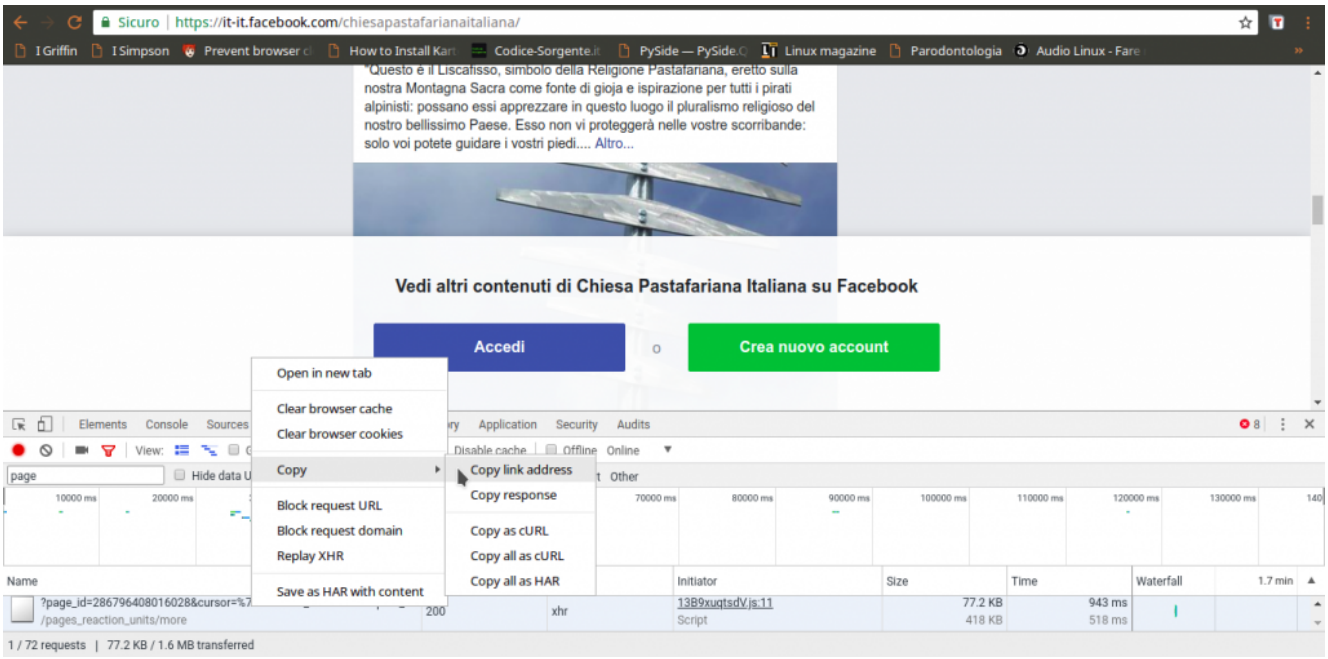
Scoprire l'indirizzo per ottenere i post

Per prima cosa dobbiamo scoprire come funzioni Facebook, cioè come vengano recuperati i vari post. In poche parole, bisogna conoscere il proprio obiettivo. Siccome si tratta di un sito web, la cosa migliore da fare è aprire una pagina Facebook (per esempio https://it-it.facebook.com/chiesapastafarianaitaliana/) col proprio browser, come Google Chrome, cliccando poi col tasto destro sulla pagina per scegliere la voce **Ispeziona**.



Tra le varie schede disponibili, quella che serve per capire cosa succeda è quella chiamata **Network**: si occupa di

presentare in tempo reale le varie richieste HTTP che vengono fatte. Siccome è ovvio che la pagina di Facebook, per caricare altri post, abbia bisogno di fare una richiesta al server di Facebook per ottenerli, è anche ovvio che apparirà qui. Tutto quello che dobbiamo fare a questo punto è scorrere la pagina verso il basso, per obbligarla a caricare altri post: nel pannello vedremo comparire una richiesta a una pagina chiamata **page_reaction_units**. Sembra proprio che abbiamo trovato ciò che ci interessava: le altre eventuali richieste sono tutte relative a file accessori, come le immagini.

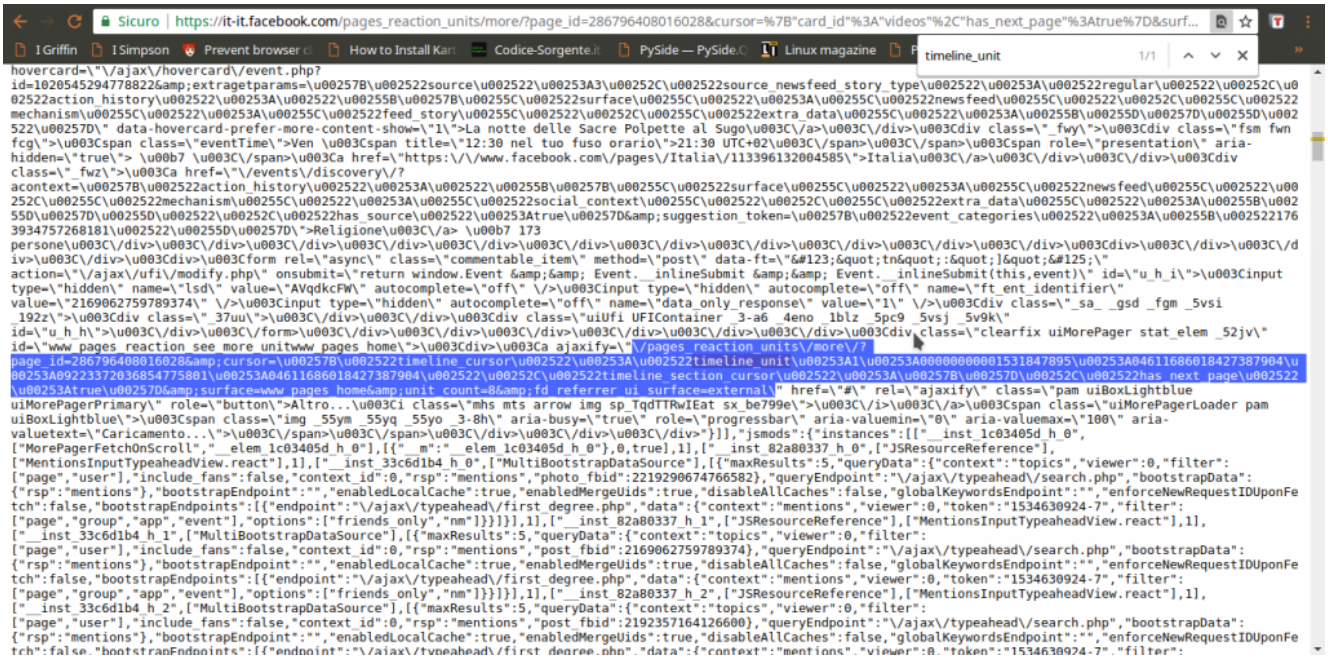


L'indirizzo della pagina contiene una serie di informazioni importanti, in particolare l'ID della pagina, ed è l'unica richiesta di questo tipo. Possiamo leggere il suo intero URL cliccandoci sopra col tasto destro del mouse e scegliendo **Copy link address**. Aprendo il link, si può capire che forma abbia la risposta: è una sorta di array JSON, una lista di oggetti vari, tra i quali il codice HTML necessario a presentare i post che sono stati richiesti. Si può facilmente distinguere il testo dei post in mezzo a tutto il codice. Alcuni caratteri vengono codificati come Unicode, inclusi alcuni pezzi dei tag HTML, e c'è sempre l'escape per i simboli /, che appaiono come \/, quindi è importante ricordarsi di convertirli in caratteri veri e propri, per poterli riconoscere facilmente (per

esempio, \u003C\/p> è in realtà

).

Guardando meglio la risposta di **pages_reaction_units** si scoprono alcune cose interessanti. Innanzitutto, tutti i post che si sono ottenuti vengono presentati tra il testo **{“__html”:** e il testo **]]}**, quindi possiamo selezionarlo facilmente. Inoltre, ogni post è preceduto dalla sua data, in vari formati. In particolare, c’è la forma Unix Time, che è molto comoda da gestire ed è sempre identificata dalla dicitura **data-utime**, quindi potremo distinguere i vari post dividendo l’intero HTML in più pezzi dopo ogni occorrenza della parola **data-utime**. E non solo: si può anche capire come ottenere ulteriori post, facendo un’altra richiesta a questa stessa pagina. Se infatti cerchiamo **pages_reaction_units** all’interno della risposta possiamo notare che c’è l’intero indirizzo di una richiesta come quella che abbiamo appena inviato, ma contenente anche il blocco **timeline_cursor**, con i riferimenti della timeline di Facebook relativi ai prossimi post. Possiamo quindi facilmente estrarre questi riferimenti per confezionare la nostra prossima richiesta.



Ora, tutto quello che è rimasto da scoprire è come costruire l'indirizzo da contattare per ottenere i vari post: sappiamo che serve **pages_reaction_units** più una serie di argomenti (che vediamo nella richiesta estrapolata dal browser). Non tutti gli argomenti sono però necessari, e possiamo scoprire quali siano superflui semplicemente provando a cancellarli uno alla volta e vedendo in quali casi si ottiene comunque il risultato desiderato. Scopriamo quindi che l'indirizzo necessario è qualcosa del tipo: **https://it-it.facebook.com/pages_reaction_units/more/?page_id=286796408016028&cursor={"timeline_cursor":"timeline_unit:1:000000000015286240 41:04611686018427387904:09223372036854775793:04611686018427387904","timeline_section_cursor":{},"has_next_page":true}&surface=www_pages_home&unit_count=8&dpr=1&__user=0&__a=1**. Di questo indirizzo, abbiamo capito che la **timeline_unit** può essere scoperta all'interno di una richiesta precedente, mentre per scoprire l'ID della pagina Facebook basta scorrere il codice HTML della pagina stessa (che si può vedere nel browser Chrome con il prefisso **view-source:**) e cercare proprio **pages_reaction_units**, e subito dopo la parola **page_id**. Giocando un po' con **unit_count** scopriamo che per la prima richiesta possiamo ottenere fino a 300 post, mentre per tutte le successive (quelle in cui si specifica la **timeline_unit**) il massimo che

si può chiedere è 8 post. Tutto il resto è solo un insieme di argomenti vari sempre uguali, che nel nostro programma potremo quindi memorizzare sotto forma di variabili.

La procedura per scaricare tutti i post sarà quindi intuitiva: si accede alla pagina leggendo il suo codice HTML per scoprire l'ID. Con questo si forma il primo URL da contattare per ottenere gli ultimi post. All'interno della risposta si prendono i post dividendoli e salvandoli separatamente, e si cercano anche i riferimenti della timeline_unit per poter fare una nuova richiesta e ottenere altri post, più vecchi. Poi si ripetono continuamente gli ultimi passaggi, leggendo la risposta, salvando i post, costruendo il nuovo indirizzo, e facendo una nuova richiesta, finché Facebook non fornisce più alcun post (il che significa che siamo arrivati all'inizio della pagina e i post sono finiti). Ora dobbiamo tradurre questa idea in uno script Python.

Leggere le pagine web

Cominciamo lo script, tutto in un unico file che chiamiamo **scrapefb.py**:

L’inizio è dato dalla shebang (#!), che su sistemi Unix è utile per automatizzare l’avvio dello script trattandolo come un eseguibile. Poi si devono importare tutte le librerie necessarie: urllib permette di scaricare il contenuto degli URL, e socket permette di stabilire un timeout sulle connessioni per chiuderle se sono inattive. Per fare il parsing della pagina web, cioè per leggere il suo contenuto distinguendo i vari “pezzi”, utilizziamo le espressioni regolari con la libreria re. Abbiamo anche bisogno di lavorare con data e ora, e accedere a funzioni relative al sistema operativo (per lettura e scrittura dei file).

Definiamo uno user agent: si tratta di una semplice stringa di testo che ogni sito richiede a chi vuole ricevere le pagine web, per capire di chi si tratti. Siccome possiamo scriverla come vogliamo, nessuno controlla davvero che stiamo dicendo la verità, possiamo scriverla in modo da convincere Facebook che il nostro script è in realtà il browser web Mozilla Firefox.

Definiamo una funzione che ci aiuti a scaricare tutto il contenuto di una pagina web, e la chiamiamo **geturl**. Prima di tutto, specifichiamo che ci serve lo useragent che abbiamo dichiarato nella sezione globale dello script. Poi ci assicuriamo di non procedere se l’url fornito alla funzione è vuoto, così evitiamo errori inutili. Costruiamo la richiesta HTTP utilizzando l’url. Servono anche un array di dati, che però in questo caso non è necessario visto che non abbiamo un form HTML da fornire alla pagina, e una intestazione. L’intestazione viene costruita con lo user agent, così Facebook ci scambierà per un browser web e non bloccherà la richiesta.

La richiesta HTTP può essere inviata usando la famosa funzione urlopen, e impostiamo anche un timeout. Il timeout è utile per non rimanere bloccati in eterno nel caso la connessione dovesse essere troppo lenta. Con un tempo di 300 secondi,

sappiamo che al massimo dopo 5 minuti la situazione verrà sbloccata. La risposta del server alla nostra richiesta può essere letta con la funzione **read**, e nel caso qualcosa non abbia funzionato impostiamo la risposta (variabile **ft**) come vuota.

In teoria potremmo tenere la risposta così com'è, ma non è una buona idea: il web è una giungla di codifiche, e se non gestiamo la cosa rischiamo di ottenere testi illeggibili. Soprattutto per Facebook, che spesso codifica le varie emoticon sotto forma di caratteri speciali Unicode. Cerchiamo quindi prima di tutto di capire se il server ci suggerisca la codifica della pagina che ci sta inviando. In caso negativo, proviamo a decodificare il testo con la classica codepage di Windows 1252, uno standard sui sistemi Microsoft precedenti a Windows 10. Se non dovesse funzionare, proviamo a decodificare tutti i caratteri usando l'utf-8 togliendo però gli slash inutili (che spesso i server web forniscono per facilitare i browser), e altrimenti cerchiamo di tradurre direttamente l'intera pagina in una stringa python. Comunque sia andata, quindi, avremo una più o meno corretta stringa python piena di tutto il codice html della pagina. Per leggere meglio il suo contenuto, utilizziamo la funzione **html.unescape** per decodificare anche le varie entità dell'html (per esempio, &gt e &lt sono rispettivamente > e <, preziosi per interpretare il codice). L'unescape delle entità html non è fondamentale, ma rende il nostro lavoro più comodo.

Cercare l'ID della pagina Facebook

Cominciamo a scrivere la funzione vera e propria per lo scraping delle pagine di Facebook. La funzione richiede, come argomenti, l'indirizzo della pagina da scaricare, la cartella in cui salvare il risultato, e se si debba salvare il

risultato come tabella CSV invece che come testo TXT.

Innanzitutto ci sono un paio di informazioni, che possiamo memorizzare in alcune variabili. Potremmo anche scriverle direttamente nelle funzioni che le usano, come vedremo, ma tenendole nelle variabili è molto più facile modificarle in futuro se dovesse essere necessario a causa di modifiche nel funzionamento di Facebook. La variabile TOSELECT_FB contiene la stringa da cercare dentro la pagina Facebook per conoscere l'URL che fornisce i vari post. Le due successive variabili sono le stringhe che delimitano l'inizio e la fine dei post nella risposta. Infatti, Facebook non fornisce solo l'elenco dei post della pagina, ma anche una serie di altre informazioni che non ci servono. Per non complicarsi la vita, bisogna avere un output pulito, quindi toglieremo tutto ciò che non ci serve isolando solo il testo presente tra quei due delimitatori. Stabiliamo poi il numero di risultati che vogliamo: il massimo consentito da Facebook (al momento) per la prima richiesta è di 300 post. Inoltre, specifichiamo un periodo di attesa prima di inviare le richieste successive, per evitare che il server possa accorgersi che ne stiamo facendo troppe tutte assieme. Le ultime due rappresentano l'inizio e la fine del link per ottenere i vari post: vedremo tra un po' come costruirlo nella sua interezza.

In questo momento siamo pronti per eseguire la prima richiesta e scaricare la pagina Facebook. L'indirizzo che contattiamo è qualcosa del tipo **https://it-it.facebook.com/chiesapastafarianaitaliana/**. Ovviamente otteniamo soltanto gli ultimi post, proprio quello che un utente normale vede quando carica la pagina. Di per se i post che appaiono non ci interessano, li otterremo contattando direttamente l'URL che fornisce tutti i post. Il codice HTML di questa pagina ci interessa soltanto perché possiamo estrarre delle informazioni. In particolare, vogliamo scoprire il dominio di Facebook, cioè tutto quello che è

compreso tra **https://** e il primo / successivo. Nel caso in esempio è **it-it.facebook.com**, ovviamente è diverso per ogni paese (una pagina spagnola non inizierà con it-it). Cerchiamo anche di capire il nome della pagina, che è tutto ciò che segue il dominio: siccome lo useremo come nome del file in cui salvare i risultati è fondamentale che non ci siano caratteri strani. Usando una espressione regolare, cancelliamo (sostituiamo con "") tutti i caratteri che non siano lettere o numeri. Fondamentale per poter proseguire è il **pageid**, cioè il numero identificativo della pagina che vogliamo scaricare: questa informazione si può recuperare dalla pagina stessa perché è sempre presente in essa un link che contiene tale numero. Il link in questione ha la forma **?page_id=286796408016028&cursor**, quindi possiamo scoprire l'ID cercando ciò che segue la parola **page_id=** e arriva fino al simbolo **&**. Ci si potrebbe chiedere come mai per cercare i vari delimitatori utilizziamo direttamente la funzione index, molto pratica e veloce, mentre per cercare la posizione del **'pages_reaction_units'**, che determina l'inizio del link in cui troviamo la pageid, usiamo le RegEx. La risposta è semplice: per ora trovare questa stringa è facile, ma in futuro potrebbe essere necessario usare una espressione regolare. In questo modo, lo script è già pronto per future modifiche.

Ora che abbiamo tutte le informazioni necessarie, possiamo costruire il nome del file in cui andremo a scrivere i post recuperati. Il nome è dato dalla cartella in cui salvare i file più **fb_** e il nome della pagina. Ovviamente, se l'utente vuole un TXT l'estensione del file sarà TXT, e se vuole un CSV l'estensione sarà CSV. Creiamo anche un altro file, con stesso nome la estensione **.tmp**. Questo è il file in cui andremo ad inserire i vari link già visitati, così se si deve riprendere lo scaricamento dei post di una pagina Facebook non lo si ricomincia da capo ogni volta, ma si riprende da dove ci si era interrotti. Per l'appunto, nel caso il file esista già vuol dire che non si deve ricominciare da capo, quindi si

carica l’intero contenuto del file in una lista, chiamata **alllinks**. In questa lista ogni elemento è un link, perché il file è stato diviso riga per riga (e quando lo scriveremo, metteremo un link in ogni riga). Definiamo anche una variabile che faccia da contatore, per sapere quante richieste di post siano state fatte, e una che stabilisca se stiamo ripristinando un download interrotto o se dobbiamo ricominciare da capo.

Richiedere i post della pagina al server di Facebook

Siamo al momento della raccolta vera e propria dei post della pagina. Siccome dobbiamo fare tante richieste una dopo l’altra, utilizziamo un ciclo. Il ciclo **while** andrà avanti finché la variabile **active** sarà True. Ne consegue che per fermare il ciclo, se necessario, non dovremo fare altro che porre tale variabile uguale a False.

Il link viene costruito unendo il dominio di Facebook, la parte iniziale del link, l’id della pagina, e la parte finale. Sarà quindi qualcosa del tipo **https://it-it.facebook.com/pages_reaction_units/more/?page_id=286796408016028&cursor={“card_id”:”videos”,”has_next_page”:true}
&surface=www_pages_home&unit_count=300&referrer&dpr=1&__user=0&__a=1**, come si può notare ci sono tutti i vari pezzi che abbiamo costruito finora. Se provate ad aprire questo indirizzo col browser vi accorgerete che fornisce una serie di informazioni, tra cui l’html dei vari post che sono stati richiesti (cioè gli ultimi 300 post della pagina). Inseriamo il link appena costruito nel file che li deve memorizzare, così se lo script dovesse bloccarsi mentre cercare di recuperare i post sapremo di dover ricominciare da questo preciso link, e non doverli rifare tutti da capo. Usando la

modalità di accesso al file “a” eseguiamo un “append”, cioè inseriamo direttamente questo link alla fine del file, in una nuova riga, senza bisogno di preoccuparci di quali altri link ci fossero prima (non dobbiamo quindi aprire il file, leggerlo, aggiungere il nuovo link, e poi salvarlo). È un risparmio di risorse importante.

Sempre utilizzando la funzione geturl possiamo recuperare anche con il nostro script tutta la risposta del server di Facebook. Siccome ci interessa soltanto la parte con i vari post che abbiamo richiesto, la estraiamo e la memorizziamo nella variabile **postshtml**. Il codice HTML dei vari post va un po’ ripulito: Facebook usa molti caratteri che non sono UTF-8 per gestire le emoticon, in genere sono utf-16. Però per il nostro scopo sono fastidiosi, le emoticon non ci interessano affatto e l’elaborazione dei testi è molto più facile con l’utf-8. Quindi ci assicuriamo di tradurre tutti i caratteri in UTF-8, togliendo anche l’escape dei caratteri speciali. Facebook, infatti, decide che alcuni caratteri sono particolari e li presenta con al loro notazione Unicode, una cosa del tipo \u0001. Questo è molto scomodo per noi, quindi forziamo la trasformazione in caratteri leggibili. A questo punto potrebbero essere rimasti dei simboli che UTF-8 non è in grado di gestire, perché si tratta delle famigerate emoticon UTF-16. Si riconoscono perché il codice Unicode è compreso tra \uD800 e \uDFFF. Siccome non ci interessano, usiamo una semplice espressione regolare per cancellarli, sostituendoli con la stringa vuota “”. Ora abbiamo finalmente l’intero codice HTML dei post, pulito e pronto per essere letto e interpretato. Siccome ogni post di Facebook è contrassegnato da un orario nel formato Unix Time (uno standard di internet), possiamo spezzare il contenuto dell’HTML nei singoli post dividendo proprio in base a **‘data-utime’**, che è la stringa che Facebook usa per indicare l’orario di un post.

In questo momento, la lista **postsarray** contiene i vari post:

in realtà, il primo elemento della lista non contiene post, perché ha tutto l'HTML precedente. Comunque, possiamo scorrere la lista e individuare i post banalmente cercando il loro timestamp, cioè l'orario della pubblicazione. È facile da identificare, perché come dicevamo ogni post viene preceduto da una span (elemento HTML) che contiene una dicitura di questo tipo: **data-utime=\"1531306802\" data-shorten=\"1\" class=\"_5ptz\">**. Siccome noi stiamo dividendo l'HTML a ogni "data-utime", è ovvio che ogni post inizierà con**=\"1531306802\" data-shorten=\"1\"…**, e quindi l'orario in formato Unix sarà il primo numero tra virgolette (nell'esempio è **1531306802**). Per essere sicuri di nona vere problemi, usiamo una RegEx per cancellare dal timestamp ottenuto qualsiasi cosa non sia un numero, e convertiamo il risultato in un **int**, cioè un numero intero. Nel caso non sia possibile risalire a questo numero, come per il primo elemento della lista che non è un vero post, consideriamo il numero pari a zero. Poi, usando datetime, possiamo convertire questo timestamp in una data facilmente leggibile, nel formato anno-mese-giorno ore:minuti:secondi. La data viene quindi aggiunta alla lista **timearray**, che abbiamo appositamente creato. Ciò significa che per ogni elemento di **postsarray**, cioè ogni post della pagina Facebook, abbiamo un corrispondente elemento di **timearray**, cioè la data della pubblicazione del post stesso.

Tutto il testo (se c'è) del post numero **i** si trova dentro all'elemento **postsarray[i]**, ma è ovviamente circondato da un sacco di altri pezzi di HTML che non ci servono. Per estrapolare soltanto il testo dei post basta prelevare tutto ciò che si trova all'interno dei paragrafi (che nella risposta di Facebook sono i tag

<\/p>). Bisogna ricordare che nello scrivere l'espressione regolare per trovare i tag il carattere \ ha bisogno di una sequenza di escape lunga, e va scritto come \\\\. La funzione **finditer** crea l'array **indexes**, che contiene tutte le posizioni in cui si trovano i vari paragrafi: un post di Facebook può

infatti essere diviso in tanti paragrafi, e noi li vogliamo tutti. Ciascun elemento di **indexes**, contiene in realtà due informazioni: la prima (cioè **0**) è l'inizio del paragrafo, e la seconda (cioè **1**) è la fine del paragrafo. Usando il classico sistema di slicing delle stringhe di Python, si può banalmente estrarre il testo di ogni paragrafo semplicemente partendo dal carattere iniziale e finale (quindi **postsarray[i][start:end]**, perché la stringa è **postsarray[i]**). Alla fine del ciclo for che legge tutti i vari **indexes**, avremo la variabile **thispost** che contiene tutti i vari paragrafi uniti, senza gli altri tag inutili.

Possiamo assegnare tutto il testo del paragrafo all'elemento stesso da cui eravamo partiti, così lo avremo ripulito. Prima, però, togliamo i tag che ancora esistono. Per esempio, il grassetto viene realizzato con i tag , quindi noi cancelliamo tutto cioè che si trova tra i simboli **<** e **>**. E cancelliamo anche gli slash non necessari. Quindi, **gatti\/cani** diventa **gatti/cani**. Alla fine presentiamo l'array sul terminale, così è più facile fare il debug e capire se qualcosa non vada bene. Lo scraping è pur sempre legato a qualcosa di molto casuale, e può capitare che in situazioni particolari qualcosa improvvisamente non funzioni.

Scoprire gli ID dei prossimi post da scaricare

Ora abbiamo ricostruito la lista **postsarray**, che contiene tutti i post presenti nell'attuale risposta di Facebook. Dobbiamo ancora capire come costruire la prossima richiesta, per ottenere una nuova risposta.

Le varie richieste che vengono inviate sono qualcosa del tipo **https://it-it.facebook.com/pages_reaction_units/more/?page_id=**

286796408016028&cursor={“timeline_cursor”:”timeline_unit:1:000 00000001528624041: 04611686018427387904:09223372036854775793: 04611686018427387904″,”timeline_section_cursor”:{}, “has_next_page”:true}&surface=www_pages_home &unit_count=8&dpr=1&__user=0&__a=1. Se ci si fa caso, è praticamente identica alla prima richiesta, con due differenze fondamentali: l’argomento **cursor** contiene i riferimenti della timeline di Facebook che indica da dove iniziano i post da scaricare. E poi c’è la **unit_count** che è limitata a **8**, quindi si possono scaricare al massimo 8 post per volta. Siccome lo stesso Facebook ha bisogno di sapere quali siano i riferimenti della timeline della pagina da scaricare, è ovvio che nella attuale risposta (quella che abbiamo appena ricevuto) ci debbano essere. E infatti ci sono, si possono trovare proprio nella forma dell’url con i due argomenti **cursor** e **unit_count**, quindi possiamo ottenerli cercando questi pezzi dell’URL dentro la stringa **newhtml** (che contiene l’ultima risposta che abbiamo ottenuto da Facebook). Siccome la prima parte dell’url è sempre la stessa, non dobbiamo fare altro che modificare la parte finale includendo i riferimenti della timeline appena ottenuti nella variabile **lending**. In questo modo, al prossimo ciclo verrà di nuovo costruito il **link**, ma usando questo **lending** come parte finale, e si potrà fare la nuova richiesta a Facebook per i successivi 8 post della pagina. Ovviamente, il testo della timeline estrapolato va un po’ pulito, con le funzioni che avevamo già visto per la rimozione dei caratteri Unicode inutili e per la decodifica dell’url. Se non riusciamo a trovare un url per i prossimi post, vuol dire che sono finiti, quindi dobbiamo interrompere il ciclo impostando la variabile **active** come falsa.

Se per qualche motivo non è stato possibile recuperare i post e le date dei post, le due apposite liste vengono inizializzate come vuote, così il programma non si bloccherà.

È arrivato il momento di salvare il risultato in un file.

L’elenco dei post scaricati durante questo ciclo è nella lista **postsarray**, possiamo trasformarla in un testo da salvare nel file prendendo i vari elementi della lista e aggiungendoli alla variabile **postsfile**, uno in ogni riga (**\n** indica un invio a capo riga). Se si desidera che il file sia un CSV, il testo del post viene preceduto dalla data di pubblicazione del post, che si trova nella lista timearray. La data e il testo del post sono separati da una tabulazione, cioè **\t**, perché se utilizzassimo altri simboli come virgola e punto virgola il risultato sarebbe inaffidabile: un post di Facebook può facilmente contenere della punteggiatura, ma non una tabulazione.

Ora che il testo da scrivere nel file è tutto nella variabile **postsfile**, dobbiamo capire se sia necessario creare il file da capo oppure no. Se questa è la prima iterazione del ciclo, e non si sta eseguendo il ripristino di un download interrotto precedentemente, bisogna scrivere il testo nel file, dunque sovrascrivendo qualsiasi cosa ci fosse (se il file esisteva già). Altrimenti, bisogna soltanto aggiungere l’attuale testo a ciò che era già stato scaricato in precedenza, usando per la modalità di scrittura append (cioè **a**) che avevamo già visto.

Ovviamente, alla fine del ciclo si incrementa di 1 il contatore **timelineiter**, che ha per l’appunto la funzione di farci sapere quante iterazioni sta facendo il programma alla ricerca di altri post da scaricare. Inoltre, prima di concludere le operazioni del ciclo, e ricominciarlo da capo, attendiamo un certo numero di secondi. Questo è importante perché se riprendessimo immediatamente a fare richieste a Facebook, il server potrebbe accorgersi che stiamo insistendo troppo e magari bloccare la connessione.

Il blocco principale dello script

Terminata la funzione per lo scraping di Facebook, ricomincia il blocco principale dello script. Per sicurezza, controlliamo che in questo momento sia stata specificamente richiesta, dall'interprete Python, la funzione main. Questo avviene soltanto se lo script è stato lanciato direttamente dal terminale, e non se è stato importato in un altro script. Questo controllo facilita un eventuale utilizzo del nostro script come libreria per altri programmi.

Ora, si definisce la pagina Facebook da cui si parte, che può essere fornita dall'utente come primo argomento dello script. Il secondo argomento, se esiste, deve rappresentare la cartella in cui si vuole ottenere il file di output, e se non è specificato si suppone che sia la cartella attuale (cioè **./**, presumibilmente la stessa in cui si trova anche lo script). Il terzo argomento, se esiste, può essere la parola "CSV": in questo caso, significa che l'utente vuole ottenere il CSV invece del TXT. Le varie informazioni vengono passate alla funzione di scraping per cominciare il download dei vari post. L'utilizzo è quindi molto semplice, e richiede praticamente soltanto l'indirizzo completo della pagina che si vuole scaricare, così come ogni utente può leggerlo in un browser web. Per esempio, si può lanciare lo script col comando **python3 scrapefb.py https://it-it.facebook.com/chiesapastafarianaitaliana/ ./ CSV** in modo da ottenere nella cartella corrente un CSV con data e testo di tutti i post della pagina della Chiesa Pastafariana Italiana, oppure si può usare il comando **python3.exe scrapefb.py https://it-it.facebook.com/chiesapastafarianaitaliana/ C:\Temp** per ottenere il TXT nella cartella **C:\Temp**.

Il codice completo

Potete trovare il codice completo dello script all'indirizzo https://gist.github.com/zorbaproject/c1f8fff28cd0becea3a0fb6d0badd159. Per utilizzarlo è necessario avere Python3 installato sul proprio sistema, ed è stato provato sia su GNU/Linux che su Windows.

Leggere, modificare, e scrivere i PDF

Tutti hanno bisogno di realizzare o modificare documenti in formato PDF: è tipicamente una delle funzioni più richieste all'interno di una applicazione qualsiasi. Soprattutto per quanto riguarda il desktop, mercato in cui i clienti principali sono aziende e pubblica amministrazione, che ovviamente utilizzano i computer per produrre documenti digitali proprio in formato PDF. Questo perché il Portable Document Format inventato da Adobe nel 1993, e le cui specifiche sono open source e libere da qualsiasi royalty, è lo standard universale ormai accettato da qualsiasi sistema operativo e su qualsiasi dispositivo per la trasmissione di documenti. Proprio perché il formato è utilizzabile gratuitamente da chiunque in lettura e scrittura, è stato inserito in praticamente qualsiasi programma ed è così conosciuto dal grande pubblico. Ormai chiunque sa cosa sia un PDF, e qualsiasi utente vorrà poter archiviare informazioni in questo formato. È quindi fondamentale essere in grado di

scrivere programmi che possano lavorare con i PDF, altrimenti si resterà sempre un passo indietro. Il problema è che il formato PDF è abbastanza complicato da gestire, ed è quindi decisamente poco pratico realizzare un proprio sistema per leggere e scrivere questi file. Bisogna basarsi su delle apposite librerie, e ne esistono varie, anche se purtroppo spesso non sono ben documentate come l'importanza dell'argomento richiederebbe, e chi si avvicina al tema rischia di non sapere da dove iniziare. Per questo motivo, abbiamo deciso di presentarvi un metodo per leggere e uno per creare PDF multipagina, con le principali caratteristiche dei PDF/A.

Come programma di esempio, abbiamo realizzato una interfaccia grafica per gli OCR Tesseract e Cuneiform, capace di funzionare sia su Windows che su GNU/Linux e MacOSX. Per motivi di spazio e di pertinenza, non presenteremo tutto il codice del programma ma soltanto le parti relative alla manipolazione dei PDF. Trovate comunque il link all'intero codice sorgente alla fine dell'articolo.

Le librerie Qt, sulle quali si basa non soltanto l'interfaccia grafica multipiattaforma del nostro programma di esempio, ma anche lo strumento di scrittura dei PDF, sono rilasciate con due licenze libere e una commerciale. Le due licenze libere sono GNU GPL e GNU LGPL: in entrambe i casi sono completamente gratuite, la differenza è che la prima richiede la pubblicazione dei programmi basati sulle Qt con la stessa licenza (quindi si deve fornire il codice sorgente), mentre la LGPL permette di utilizzare le librerie pur non distribuendo il codice sorgente del proprio programma. L'opzione GPL è valida per tutte le applicazioni che verranno rilasciate come software libero da programmatori amatoriali, mentre la LGPL è più indicata per aziende che non vogliono rilasciare il

programma come free software. La licenza commerciale serve solo nel caso si voglia modificare il codice sorgente delle librerie Qt stesse senza pubblicare il codice delle modifiche.

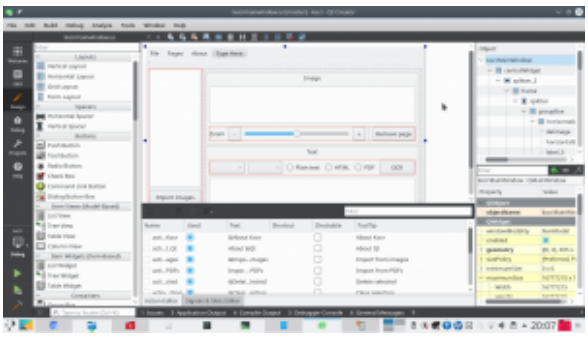
Il programma è scritto in C++ con le librerie multipiattaforma Qt, delle quali ci serviremo per scrivere i PDF usando le funzioni della classe QPDFWriter. Per la lettura dei PDF, invece, utilizzeremo la libreria libera e open source Poppler, che si integra perfettamente con le librerie Qt.

Come funziona il formato PDF?

Il formato PDF è uno standard ufficiale dal 2007, declinato in una serie di sottoformati: A,X,E,H,UA, a seconda dei vari utilizzi che se ne vogliono fare. Quello che si segue solitamente è il PDF/A, progettato per l'archiviazione dei documenti anche a lungo termine: è pensato per integrare tutti i componenti necessari. Prima che si stabilisse questo standard, infatti, i PDF non erano davvero adatti a conservare e trasmettere documenti, perché mancavano spesso alcuni componenti fondamentali. Per esempio, se un PDF veniva visualizzato su un computer nel quale non erano installati i font con cui sul PC originale era stato scritto il testo, tutta l'impaginazione saltava. Ora, invece, i font possono essere integrati, assieme ad eventuali altri oggetti, così è possibile visualizzare correttamente un PDF/A su qualsiasi dispositivo, a prescindere dal suo sistema operativo. Questo significa che ogni PDF moderno è di fatto un po' più grande di quanto lo sarebbe stato un PDF degli anni '90, perché porta al suo interno i vari font, ma questo non è un problema considerando che il costo dello spazio dei dischi rigidi diminuisce continuamente e un paio di kilobyte in più in un file non si notano nemmeno.

Il formato PDF nasce da un formato precedente che è tutt'ora

in uso e che si chiama PostScript. PostScript è di fatto un linguaggio di programmazione che permette di descrivere delle pagine: i file PS sono dei semplici file di testo che contengono una serie di istruzioni per il disegno di una pagina, con le sue immagini e il testo. Si tratta di un linguaggio che va interpretato, quindi la sua elaborazione richiede una buona quantità di risorse e di tempo. Un file PDF, invece, è di fatto una sorta di PS già interpretato, il che permette di risparmiare tempo. Per fare un esempio, in un file PS si troveranno molte condizioni “if” e cicli “loop”, e si tratta di istruzioni che consumano molte risorse quando vanno interpretate. Nei PDF, invece, viene direttamente inserito il risultato dei vari cicli, così da risparmiare tempo durante la visualizzazione. Quello che è importante capire è che il formato PDF è progettato per la stampa, è pensato per essere facilmente visualizzato e stampato allo stesso modo su qualsiasi dispositivo. Insomma, una funzione di sola lettura. Non è affatto progettato per permettere la continua modifica dei file. Ciò non significa che sia proibito, i file PDF possono ovviamente essere modificati come qualsiasi altro file, ma la modifica può essere molto complicata da fare in certi casi proprio perché le informazioni vengono memorizzate puntando a massimizzare l’efficienza della lettura, non della scrittura o della modifica. Per esempio, i testi vengono memorizzati una riga alla volta, e non in blocchi di paragrafi o colonne, come invece risulterebbe comodo per modificarli successivamente. Un’altra differenza importante è che nei PDF ogni pagina è un elemento a se stante, mentre nei PostScript le pagine sono legate e condividono alcune caratteristiche (come le dimensioni).



Utilizzando l'IDE gratuito QtCreator è molto facile anche disegnare l'interfaccia grafica multipiattaforma

Includere Poppler

Cominciamo subito col nostro programma di esempio. Le librerie necessarie possono essere incluse nell'intestazione del codice come da prassi del C++. Quelle che servono per la gestione dei PDF sono le seguenti:

Per scrivere i PDF utilizzeremo infatti la libreria QpdfWriter, che si trova nella stessa cartella di tutte le altre librerie Qt, e che quindi viene trovata in automatico dall'IDE. Per la lettura dei PDF, invece, useremo Poppler, che va installata a parte. Qui le cose cambiano un po', perché mentre in GNU/Linux esiste un percorso standard nel quale installare le librerie, e quindi si può facilmente trovare poppler nella cartella **poppler/qt5/**, su Windows questo non esiste. Quindi, sfruttando gli **ifdefine** forniti dalle librerie Qt, possiamo distinguere la posizione dei file che contengono la libreria Poppler a seconda del fatto che il sistema sia Windows (**Q_OS_WIN**) o GNU/Linux (**Q_OS_LINUX**). La posizione delle librerie per Windows potrà essere stabilita nel file di progetto, che vedremo più avanti. Possiamo ora cominciare a vedere il codice: non lo vedremo tutto, solo le parti

fondamentali per la gestione dei PDF.

Entro breve, le librerie Qt integreranno direttamente una classe per la lettura dei PDF, chiamata QPDFDocument, senza quindi la necessità di usare Poppler. Al momento tale classe non è ancora considerata stabile, quindi abbiamo deciso di presentare questo articolo basandoci ancora su Poppler. Quando il rilascio di QtPdf sarà ufficiale, la presenteremo in nuovo articolo.

La prima funzione che implementiamo dovrà permettere l'importazione dei PDF. Infatti, vogliamo permettere agli utenti di importare dei PDF scansionati, in modo da poter eseguire su di essi l'OCR e ricavare il testo.

Chiamando la funzione **getOpenFileNames** di **QFileDialog** si visualizza una finestra standard per consentire all'utente la selezione di più file, il cui percorso completo viene inserito in una lista di stringhe che chiamiamo **files**.

Possiamo anche creare una MessageBox per chiede conferma all'utente, così se dovesse avere scelto i file per sbaglio potrà annullare il procedimento prima di cominciare a lavorare sui file (operazione che può richiedere del tempo).

Banalmente, se il pulsante premuto dall'utente è **Cancel**, allora interrompiamo la funzione. Altrimenti, con un semplice ciclo for scorriamo tutti gli elementi della lista di file, passandoli uno alla volta a un funzione che si occuperà di estrarre le pagine dal PDF e aggiungerle alla lista delle pagine su cui lavorare.

Aprire un PDF in lettura

Abbiamo chiamato la funzione che opera effettivamente l'estrazione delle pagine da un PDF, **addpdftolist**.

Questa funzione comincia controllando che il file che ha ricevuto come argomento **pdfin** sia esistente (**QfileInfo.exists** controlla che il file esista e non sia vuoto).

Ora abbiamo bisogno di una cartella temporanea, nella quale inserire tutte le immagini che estrarremo dalle pagine del PDF. La libreria **QTemporaryDir** si occupa proprio di creare una cartella temporanea a prescindere dal sistema operativo. Possiamo memorizzare il percorso di tale cartella in una stringa che chiamiamo **tmpdir**. Dobbiamo anche specificare che la cartella non va sottoposta all'auto rimozione, altrimenti il programma cancellerà la cartella automaticamente al termine di questa funzione, mentre noi ne avremo ancora bisogno in altre funzioni. La cancellazione di tale cartella potrà essere fatta manualmente alla chiusura definitiva del programma.

Siamo finalmente pronti per leggere il PDF. Basta creare un oggetto di tipo **Poppler::Document**, usando la funzione load che permette per l'appunto la lettura di un file PDF. Se il PDF non conteneva un documento valido, conviene terminare la funzione con l'istruzione return per evitare problemi.

Il documento potrebbe avere più pagine, quindi utilizziamo un ciclo for per leggerle tutte una alla volta.

Ogni pagina può essere estratta usando un oggetto **Poppler::Page**, e con l'apposita funzione **page** di un documento. Se la pagina è invalida, il ciclo si ferma.

La pagina può poi essere renderizzata in una immagine, rappresentata dall'oggetto **QImage**, secondo una carta risoluzione orizzontale e verticale (che di solito coincidono, ma non sempre).

Poppler permette di leggere tutti i componenti di un PDF, annotazioni incluse

Nel nostro programma, consideriamo tale risoluzione pari a 300 dpi, e l'abbiamo inserita in una apposita variabile all'inizio del programma chiamata per l'appunto **dpi**.

Per salvare l'immagine della pagina basta chiamare la funzione **save** dell'immagine. Tuttavia, prima dobbiamo decidere il nome del file: sarà ovviamente composto dal percorso della cartella temporanea più il nome **tmppage** seguito dal numero progressivo della pagina e l'estensione **tiff**. Il numero della pagina viene scritto con 4 cifre, giustificando con lo 0. Quindi, la pagina 1 sarà **tmppage0001**, mentre la pagina 23 sarà **tmppage0023**, e la 145 sarà **tmppage0145**. In questo modo siamo sicuri di non confondere mai l'ordine delle pagine.

È bene ricordarsi di eliminare l'oggetto pagina e il documento, per liberare la memoria (lavorando ad alte risoluzioni è facile che venga richiesta molta RAM per svolgere queste operazioni).

Avremmo potuto inserire direttamente ogni immagine estratta nell'elenco delle immagini che vogliamo passare all'OCR, ma

possiamo anche semplicemente leggere il contenuto della cartella temporanea cercando tutti i file che contiene e, scorrendoli uno ad uno, passare il loro percorso alla funzione **addimagestolist**. È infatti questa la funzione che si occuperà di inserire le singole immagini nella lista. Chiamando la funzione soltanto dopo l'estrazione delle pagine, le immagini appariranno nell'interfaccia grafica tutte assieme e l'utente capirà che la procedura è terminata.

La funzione in questione è molto semplice: viene creato un nuovo elemento del qlistwidget (l'oggetto che nell'interfaccia grafica del nostro programma funge da elenco delle pagine). All'elemento viene assegnata una icona, che proviene dal file stesso e che quindi costituirà la sua anteprima. L'elemento viene infine aggiunto all'oggetto presente nell'interfaccia grafica (**ui**).

Il file di progetto

Per consentire al compilatore di trovare la libreria Poppler basta inserire nel file di progetto (.pro) le seguenti righe:

In questo modo il compilatore saprà che su Windows i file .h si troveranno nella cartella **include/poppler-qt5** del codice sorgente, mentre la libreria compilata sarà nella cartella **lib**.

Fusione dei PDF

Dopo avere eseguito l'OCR sulle varie pagine, si ottengono da Tesseract tanti PDF quante sono per l'appunto le pagine del documento. Ciò significa che dovremo riunirle manualmente,

fondendo assieme tutti i vari file in un unico PDF. Per farlo, prima di tutto decidiamo il nome di un file temporaneo nel quale riunire tutti i PDF:

Lo facciamo sfruttando lo stesso meccanismo che abbiamo usato per la cartella temporanea, ma con la libreria **QTemporaryFile**. Ovviamente, il file dovrà avere estensione **pdf**, e il suo nome è contenuto nella variabile **tmpfilename**.

Per scrivere sul PDF temporaneo, basta creare un nuovo oggetto di tipo QpdfWriter associato al file e un oggetto **Qpainter** associato al **pdfWriter**. Il QPainter è il disegnatore che si occuperà di, per l'appunto, disegnare il contenuto del PDF secondo le nostre indicazioni.

Le varie pagine, cioè i pdf da riunire, si trovano nella stringa **allpages** separati dal simbolo **|**. Con un semplice ciclo for possiamo prendere un pdf alla volta, inserendo il suo nome nella stringa **inp**.

Se quello su cui stiamo lavorando non è il primo dei file da unire (quindi il contatore delle pagine **i** è maggiore di 0), allora possiamo inserire una interruzione di pagina nel PDF finale con la funzione **newPage**. Questo ci permette di unire i vari file dedicando una nuova pagina a ciascuno.

Possiamo quindi aprire il pdf usando, come già visto, **Poppler::Document**. Con un ciclo for scorriamo le varie pagine: ciascuno dei file da unire dovrebbe contenere una sola pagina, ma è comunque più prudente usare un ciclo per non correre rischi.

Le varie TextBox

Ora dobbiamo estrarre il testo della pagina, cioè il testo che Tesseract ha inserito grazie alla funzione di OCR.

Potremmo semplicemente prelevare il testo con la funzione **text**, ma preferiamo usare **textList**. Infatti, la prima ci fornisce semplicemente tutto il testo della pagina, ma a noi questo non va bene: abbiamo bisogno di avere anche l'esatta posizione, nella pagina, di ogni parola. Per questo esiste **textList**, una lista di **Poppler::TextBox**, dei rettangoli che contengono il testo e hanno una precisa posizione e dimensione.

Con un ulteriore ciclo for possiamo scorrere tutte le **textBox** ottenendo il rettangolo (**QrectF** è un rettangolo con dimensioni float) che le rappresenta usando la funzione **boundingBox**.

Ora c'è un piccolo problema: le dimensioni e la posizione del rettangolo sono state indicate, da Poppler, con il sistema di riferimento della pagina che stiamo leggendo. Invece, il nostro pdfWriter avrà probabilmente un sistema di riferimento diverso, a causa della risoluzione. Possiamo calcolare il rapporto orizzontale e verticale semplicemente dividendo larghezza e altezza della pagina di pdfWriter per quelle della pagina di Poppler.

Adesso possiamo tranquillamente scrivere il testo usando il nuovo rettangolo, che abbiamo appena calcolato, come riferimento. Il testo (attributo **text** della **textBox** attuale) si aggiunge usando la funzione drawText del **painter**.

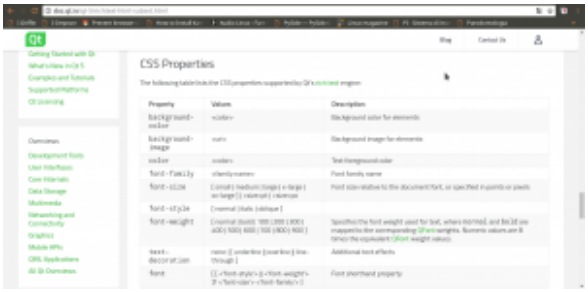
Soltanto dopo avere terminato questo ciclo for, e quindi avere scritto tutti i testi dove necessario, possiamo disegnare

sulla pagina l'immagine di sfondo, con la funzione drawPixmap che si usa per inserire in un **painter** una immagine a mappa di pixel (una bitmap qualsiasi). La pixmap è ovviamente ottenuta dall'immagine che preleviamo tramite Poppler usando la già vista funzione **renderToImage**. Inserendo l'immagine dopo il testo, siamo sicuri che sarà visibile soltanto l'immagine, e il testo risulterà invisibile ma ovviamente selezionabile e ricercabile. In alternativa avremmo anche potuto scegliere il colore "trasparente" per il testo.

Ovviamente, quando abbiamo finito di leggere un file, dobbiamo eliminare il suo oggetto **document** per non occupare troppo spazio. Per quanto riguarda il PDF che stiamo scrivendo, non c'è bisogno di chiudere il file: QpdfWriter lo farà automaticamente appena la funzione termina.

Scrivere dell'HTML

C'è ancora un ultimo caso da considerare: se invece di Tesseract si vuole utilizzare l'OCR Cuneiform su Windows, purtroppo non si ottiene un PDF e nemmeno un file HOCR (cioè un HTML con la posizione delle varie parole). Si ottiene soltanto un semplice file HTML, che mantiene la formattazione ma non la posizione delle parole.



Un testo formattato può essere inserito in un PDF con QTextDocument usando la formattazione CSS delle pagine HTML

(http://doc.qt.io/qt-5/richtext-html-subset.html)

Non è ottimale, ma può comunque essere utile avere un PDF che contenga il testo nella pagina, così lo si può ricercare facilmente. In questo caso, la prima cosa da fare è leggere il file html che si ottiene:

Leggendo il file come semplice testo grazie alle librerie QFile e QTextStream, possiamo inserire tutto il codice nella stringa **hocr**.

Ora, possiamo creare un nuovo documento di testo formattato, usando la libreria QTextDocument. Il contenuto del testo sarà indicato proprio dal codice **html** della stringa hocr, che quindi mantiene la formattazione. Impostiamo anche la larghezza massima del testo pari a quella della pagina di **pdfWriter**.

Come prima, dovremo calcolare la corretta dimensione con cui inserire il testo, per evitare che sia troppo piccolo o troppo grande. Siccome stavolta è solo testo, possiamo calcolare la dimensione del font con cui scriverlo usando una proporzione.

Dopo avere scelto la giusta dimensione del testo affinché riempia tutta la pagina, possiamo inserire il testo nel painter, e quindi nel PDF, usando la funzione drawContents del QTextDocument. Il vantaggio d i questa funzione, rispetto a drawText, è che in questo modo si mantiene la formattazione e l'allineamento standard HTML.

Ovviamente, anche in questo caso si conclude la pagina inserendo sopra al testo l'immagine della pagina stessa, così il testo non sarà visibile, ma comunque ricercabile e selezionabile.

Il codice sorgente e il binario dell'esempio

Per capire come venga organizzato il codice sorgente, vi conviene controllare quello del nostro programma di esempio. Banalmente, il programma è composto da un file di progetto, un file **main.cpp** che costituisce la base dell'eseguibile, e due file (uno .h e uno .cpp) per la classe **mainwindow**, che rappresenta l'interfaccia principale del programma. Inoltre, abbiamo inserito due cartelle con il codice sorgente e il codice binario della libreria Poppler per Windows.

Trovate tutto il codice su GitHub assieme a dei pacchetti precompilati per Windows e GNU/Linux: **https://github.com/zorbaproject/kocr/releases**

IBM e il quantum computing per tutti

Due anni fa è avvenuto, un po' in sordina, uno di quegli eventi che potrà avere ripercussioni a lungo termine sullo sviluppo dell'informatica: IBM ha aperto l'accesso, tramite cloud computing, al proprio calcolatore quantistico. Si tratta di qualcosa di cui tutti, anche i meno esperti, hanno sentito parlare, ma spesso si fa molta confusione sull'argomento e non si riescono a comprendere appieno le implicazioni di questa rivoluzione. Naturalmente, per capirle dobbiamo prima capire la differenza tra un computer classico ed uno quantistico. Prima di cominciare, però, specifichiamo un punto importante:

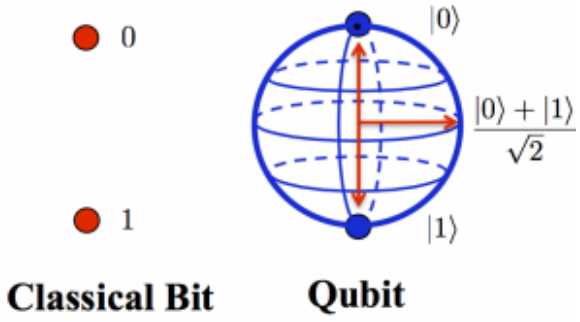
la rivoluzione non è ancora iniziata, la si sta soltanto preparando, per ora. Con l'attuale computer quantistico di IBM non si può fare molto, e le normali operazioni di programmazione sembrano inutilmente complicate. Questo perché la potenza di calcolo è ancora troppo bassa. Naturalmente, il problema dei computer quantistici è che man mano che la loro potenza (o, se volete, il numero di qubit, lo spiegheremo più avanti) aumenta diventano molto più difficili e costosi da costruire. Ed è per questo motivo che IBM permette l'accesso a tutti i programmatori tramite il cloud: in questo modo potrà contare su una squadra di alfa-tester volontari, e potrà cercare di migliorare l'efficienza del calcolatore scoprendo i problemi che salteranno fuori durante i vari test. Insomma, i calcolatori quantistici per ora non sono utili. Ma tra qualche anno potrebbero esplodere con la loro potenza, e potrebbero consentirci di eseguire algoritmi che oggi con un computer classico non si possono proprio realizzare.

Le macchine di Turing

Un computer è fondamentalmente una macchina di Turing. Una macchina di Turing è un modello matematico che rappresenta ciò che un calcolatore programmabile può fare. Ovviamente, ciò che un calcolatore può fare nel mondo reale è dettato dalle leggi della fisica classica. La macchina di Turing nasce per rispondere ad una domanda: esiste sempre un metodo attraverso cui un qualsiasi enunciato matematico possa essere stabilito come vero o falso? In altre parole, è possibile sviluppare un algoritmo con cui capire se una qualunque affermazione sia vera o falsa?

La risposta è no, non è possibile sviluppare un algoritmo generale che stabilisca la veridicità di una qualsiasi affermazione, e questo perché in realtà il concetto di "algoritmo" non è ben definito. Tuttavia, Alan Turing propose comunque un modello matematico (la sua famosa "macchina") con

cui verificare, per ogni singola affermazione, se sia possibile arrivare a stabilirla come vera o falsa. Con una macchina di Turing infatti è possibile sviluppare algoritmi per ridurre una affermazione ai suoi componenti di base, cercando di verificarla, e vi sono due opzioni: o la macchina ad un certo punto termina l'algoritmo e fornisce una risposta, oppure l'algoritmo andrà avanti all'infinito (in una sorta di loop continuo, chiamato **halting**) senza mai fornire una risposta. Oggi si usa proprio la macchina di Turing per definire il concetto di "algoritmo".



Mentre un bit ha due soli possibili valori, un qubit ha una probabilità di essere più vicino a 1 o a 0, quindi è un qualsiasi numero compreso tra 0 e 1

Una macchina di Turing utilizza come input-output un nastro su cui legge e scrive un valore alla volta, in una precisa cella del nastro, ed in ogni istante di tempo **t(n)** la macchina avrà un preciso stato **s(n)** che è il risultato dell'elaborazione. Ovviamente la macchina può eseguire alcune operazioni fondamentali: spostarsi di una cella avanti, spostarsi di una cella indietro, scrivere un simbolo nella casella attuale, cancellare il simbolo presente nella casella attuale, e fermarsi.

Un modello così rigido, che esegue operazioni elementari per tutto il tempo che si ritiene necessario, può di fatto

svolgere qualsiasi tipo di calcolo concepibile. Tuttavia, così come Godel aveva dimostrato che alcuni teoremi non si possono dimostrare senza cadere in un ciclo infinito, anche con la macchina di Turing non è detto che arrivi ad un risultato, perché alcuni calcoli non si possono portare a termine senza cadere in un processo infinito (in altre parole, il tempo necessario per la soluzione sarebbe infinito, e nel mondo reale nessuno di noi può aspettare fino all'infinito per avere una risposta). Per fare un esempio banale, se ciò che vogliamo fare è semplicemente ottenere il risultato di 3*4, è ovvio che basta scomporre il problema nei suoi termini fondamentali, ovvero un ciclo ripetuto 4 volte in cui si somma +3. Il ciclo richiede un numero finito di passaggi, quindi la moltiplicazione 3*4 è una operazione che può essere svolta con una macchina di Turing senza cadere nell'**halting**. Se volessimo moltiplicare due numeri enormi servirebbe un ciclo con molti più passaggi, ma sarebbero comunque un numero finito, quindi anche se potrebbe essere necessario un tempo molto lungo, prima o poi il ciclo finirebbe e la macchina produrrebbe un risultato.

Dalla fisica classica alla fisica quantistica

Ora, abbiamo detto che per la macchina di Turing valgono le leggi della fisica classica, e ci riferiamo in particolare ai principi della termodinamica. Il primo principio stabilisce quali eventi siano possibili e quali no, il secondo principio stabilisce quali eventi siano probabili e quali no. Il primo principio, espresso come definizione dell'energia interna di un sistema (U):

$dU=Q-L$

ci dice che l'energia interna di un sistema chiuso non si crea e non si distrugge, ma si trasforma ed il suo valore rimane

dunque costante. Infatti, in un sistema isolato Q=0 (il calore) quindi dU=-L, ovvero l'energia interna può trasformarsi in lavoro e viceversa senza però sparire magicamente. Il secondo principio, che viene espresso con l'equazione

$dS/dt>=0$

ci dice che ogni evento si muove sempre nella direzione che fa aumentare l'entropia e mai nell'altra. Al massimo può rimanere costante, nelle rare situazioni reversibili. Infatti la derivata dell'entropia (S) in funzione del tempo (t) è sempre maggiore o uguale a zero. In altre parole, l'enunciato del secondo principio della termodinamica si può riassumere con la frase "riscaldando un acquario si ottiene una zuppa di pesce, ma raffreddando una zuppa di pesce non si ottiene un acquario". Insomma, la maggioranza degli eventi termodinamici non è reversibile, almeno nel mondo reale, e questo vale anche per la macchina di Turing.

Una macchina di Turing quantistica, invece, si baserebbe sulle leggi fisiche della meccanica quantistica, e quindi le sue operazioni potrebbero essere reversibili. Il vantaggio ovvio è che se le operazioni sono reversibili di fatto non è possibile che la macchina quantistica cada nell'**halting** della macchine di Turing classiche. Basandosi proprio sulla meccanica quantistica, ci saranno una serie di differenze con la macchina di Turing classica:

- un bit quantistico, chiamato **qubit**, non può essere letto con assoluta precisione. In genere, si fa soltanto una previsione probabilistica del fatto che sia più vicino a 0 o a 1
- la lettura di un qubit è una operazione che lo danneggia modificandone lo stato, quindi non è più possibile leggerlo nuovamente
- date le due affermazioni precedenti, è ovvio che non si possano conoscere con precisione le condizioni iniziali

e nemmeno i risultati
- un qubit può essere spostato da un punto all'altro con precisione assoluta, a condizione che l'originale venga distrutto: è il teletrasporto quantistico
- i qubit non possono essere scritti direttamente, ma si possono scrivere sfruttando l'entanglement, cioè la correlazione quantistica
- un qubit può essere 0 od 1, ma può anche essere una combinazione di questi due stati (un po' 0, un po' 1, come il gatto di Schroedinger), quindi può di fatto contenere molte più informazioni di un normale bit

Apparentemente, queste caratteristiche rendono la macchina inutile, ma è solo il punto di vista di una persona abituata a ragionare nel modo tradizionale. In realtà, una macchina di Turing quantistica è imprecisa durante il momento di input e quello di output, ma è infinitamente precisa durante i passaggi intermedi.

Per chi non ha studiato le basi della fisica, uno sviluppatore ha scritto una guida che riassume alcuni dei concetti fondamentali, soprattutto per quanto riguarda le porte logiche, fondamentali per manipolare i qubit.

I qubit possono essere implementati misurando la proprietà di spin dei nuclei degli atomi di alcune molecole, oppure la polarizzazione dei fotoni (anche i fotografi dilettanti sanno che la luce, composta da fotoni, può essere polarizzata usando appositi filtri). In realtà, pensandoci bene, ci si può accorgere che una macchina di Turing quantistica non è altro che una macchina di Turing il cui nastro di lettura/scrittura contiene valori casuali. Qual è il vantaggio della casualità? Semplice: la possibilità di fare tentativi a caso per cercare la soluzione di un problema che non si riesce ad affrontare

con un algoritmo tradizionale. Naturalmente, aiutando un po' il caso con un algoritmo.

Il vantaggio della casualità "vera"

Nei casi degli algoritmi più semplici, questa idea è solo una complicazione, ma in algoritmi molto complessi e lenti la macchina quantistica può fornire un risultato accettabile in tempi molto ridotti. Possiamo fare un paragone con il metodo Monte Carlo, un metodo per ottenere un risultato approssimato sfruttando tentativi casuali. Facciamo un esempio: immaginiamo di avere una sagoma disegnata su un muro e di voler misurare la sua area. Se la sagoma è regolare, come un cerchio, è facile: basta usare l'apposito algoritmo (per esempio "raggio al quadrato per pi greco"). Ma se la sagoma è molto più complicata, come la silhouette di un albero, sviluppare un apposito algoritmo diventa estremamente complicato e lento. Il metodo Monte Carlo ci suggerisce di prendere un fucile mitragliatore e cominciare a sparare a caso contro il muro: dopo un po' di tempo non dovremo fare altro che contare il numero di proiettili che sono finiti dentro alla sagoma e moltiplicare tale numero per la superficie di un singolo proiettile (facile da calcolare sulla base del calibro). Otterremo una buona approssimazione della superficie della sagoma: la qualità dell'approssimazione dipende dal numero di proiettili abbiamo sparato col fucile ma, comunque vada, il tempo complessivo per ottenere il valore della superficie è decisamente poco rispetto all'uso di un algoritmo metodico. Un metodo simile che si usa molto nell'informatica moderna è rappresentato dagli algoritmi genetici, i quali hanno però tre svantaggi: uno è che deve essere possibile sviluppare una funzione di fitness, cosa non sempre possibile (per esempio non si può fare nel brute force di una password), e l'altro è che comunque verrebbero eseguiti su una macchina che si

comporta in modo classico, quindi il sistema sarebbe comunque poco efficiente (pur offrendo qualche vantaggio in termini di tempistica). Inoltre, è praticamente impossibile ottenere delle mutazioni davvero casuali nei codici genetici che si utilizzano per cercare una soluzione: nei computer classici la casualità non esiste, è solo una illusione prodotta da qualche algoritmo che a sua volta non è casuale. I qubit risolvono questi problemi, visto che sono rappresentati da oggetti fisici che sono naturalmente casuali.

Scomposizione in fattori primi

Ovviamente, uno degli utilizzi più interessanti di una macchina di Turing quantistica è l'esecuzione di un **algoritmo di fattorizzazione di Shor**. Come si impara a scuola, fattorizzare un numero (cioè scomporlo nei fattori primi) è una operazione che richiede molto tempo, sempre di più man mano che il numero diventa grande. Si tratta di una cosa molto importante perché l'RSA, la crittografia che al momento protegge qualsiasi tipo di comunicazione (dai messaggi privati alle transazioni finanziarie) si basa proprio sui numeri primi e la sua sicurezza è dovuta al fatto che con un normale computer, ovvero una macchina di Turing classica, scomporre in fattori primi un numero sia una operazione talmente lenta che sarebbero necessari degli anni per riuscirci. Ma sfruttando un calcolatore quantistico e l'algoritmo di fattorizzazione di Shor, questa operazione diventa "facile" e la si può svolgere in un tempo che si calcola con un polinomio, dunque è un tempo "finito" invece di "infinito" e solitamente abbastanza breve. Con questo algoritmo, si potrebbero calcolare le chiavi private di cifratura di tutti i messaggi più riservati, e l'intera sicurezza delle telecomunicazioni crollerebbe. Al momento, con gli attuali computer quantistici, non è possibile applicare l'algoritmo di Shor in modo generale, ma sono già

state realizzate alcuni versioni semplificate dell'algoritmo adattate per specifici casi. Se i computer quantistici diventassero più potenti ed affidabili, diventerebbe possibile sfruttare l'algoritmo su qualsiasi chiave crittografica e far crollare la sicurezza di internet. Il mondo, comunque, non tornerebbe all'età della pietra: sempre usando i computer quantistici sarebbe possibile utilizzare un sistema di crittografia a "blocco monouso" con distribuzione quantistica della chiave crittografica, l'unico metodo di cifratura che sarebbe davvero inviolabile. Nel senso che affinché si possa forzare la crittografia con chiave quantistica le leggi della fisica dovrebbero essere sbagliate (e non lo sono).

Il quantum computer IBM è gratuito?

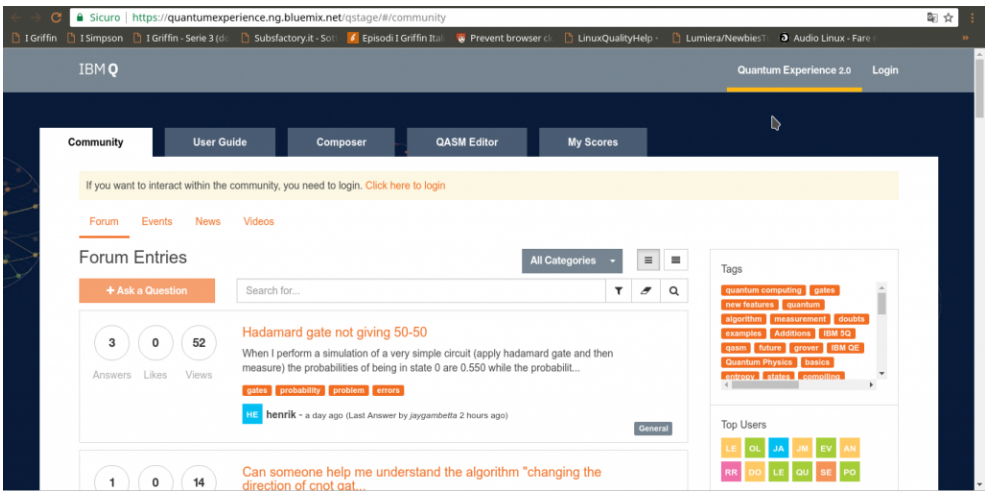
Al momento in cui scriviamo, l'accesso al calcolatore quantistico di IBM è gratuito. Quando ci si registra, si ottengono una decina di "units", cioè dei crediti virtuali. L'esecuzione di un programma costa intorno alle 3 units. Quando le proprie units sono terminate, dopo 24 ore IBM ricarica automaticamente le units iniziali, così si può continuare a sperimentare.

Un esempio pratico: OpenQASM

Il computer quantistico di IBM ha 5 qubit, ma per la maggioranza degli algoritmi sviluppati finora se ne usano soltanto due. Una operazione che può essere interessante provare è la trasformata di Fourier: per chi non la conoscesse, si tratta di una trasformazione che permette di "semplificare" una funzione algebrica riducendone il rumore e rimuovendo i dati non interessanti. Siccome la sua

implementazione in un calcolatore classico può essere lunga, è stato sviluppato un algoritmo chiamato FFT, cioè trasformata di Fourier veloce. Per implementare una trasformata di Fourier nel computer quantistico IBM si usa il suo linguaggio nativo OpenQASM:

La sintassi del linguaggio è una via di mezzo tra assembly e C: **qreg** crea un registro di bit quantistici (4, nell'esempio) mentre **creg** crea un registro di bit classici (sempre 4). Poi si possono applicare dei **gate**, ovvero delle porte logiche ai qubit. Per esempio, applicando il gate **X** ai qubit 0 e 2, il registro q diventerà 1010, perché questa porta logica non fa altro che invertire il valore di un qubit (che di default è sempre 0). Il comando **barrier** impedisce che avvengano trasformazioni nei qubit. Altri tipi di porte logiche sono **H** e **CU1**. La prima serve a produrre una variazione casuale nella probabilità di un qubit, cioè nella probabilità che esso sia più vicino ad essere 0 oppure 1. Eseguendo il gate H su tutti i qubit, ci si assicura che i loro stati siano davvero casuali. La seconda, invece, è una delle porte logiche fornite da IBM (ce ne sono una dozzina), che permette di eseguire i passaggi per la serie di Fourier. Infine, il comando **measure** traduce i qubit in bit riempiendo il registro classico **c**, i cui valori possono quindi essere letti senza problemi.



Sul sito https://quantumexperience.ng.bluemix.net/ è

possibile iscriversi usando il proprio account GitHub o Google

Naturalmente, grazie alla casualità, se si esegue l'algoritmo più volte si otterranno risultati diversi. Tuttavia, con questo algoritmo si può approssimare in un tempo molto breve una trasformata di Fourier per l'array classico 1010. Ovviamente, non si può davvero utilizzare questo algoritmo per analizzare il segnale di un ricevitore radio, non sarebbe affatto pratico. Però in futuro potremmo riuscire ad avere computer quantistici tanto potenti da permetterci di eseguire una trasformata di Fourier, approssimata, in tempi rapidi anche per quantità di dati enormi. Intanto, possiamo provare a giocarci un po' e a inventare nuovi gate, nuove porte logiche per i qubit. In fondo, il motivo per cui IBM ha reso pubblico l'accesso al calcolatore quantistico è che soltanto sperimentando nuovi utilizzi di questo tipo di computer sarà possibile aumentare l'efficienza e soprattutto capire quanto davvero il qubit rivoluzionerà la storia dell'informatica e la vita di tutti i giorni.

Caricare codice QASM con Python

Per caricare un programma sul cloud di IBM ed eseguirlo con il processore quantistico si può installare l'apposita libreria direttamente con il comando:

Poi si può sfruttare il codice di test per provare il caricamento di un codice OpenQASM. Prima di tutto si deve modificare il file **config.py** inserendo il proprio token di autorizzazione personale:

e poi si può avviare il programma. Il suo codice è abbastanza

semplice, ne presentiamo i punti salienti:

Il programma comincia importando la libreria di IBM ed il file **config** che contiene il token.

Le varie funzione del programma sono inserite in una apposita classe, chiamata **TestQX**, per sfruttare la programmazione ad oggetti dalla routine principale.

Una delle prime funzioni è quella che si occupa della verifica del token: viene creato l'oggetto api, dal quale si possono ottenere le credenziali utilizzando con il metodo **_check_credentials()**. La funzione restituisce un valore true se le credenziali sono valide.

Un'altra funzione, sfruttando il metodo **get_last_codes()** permette di verificare se siano già stati caricati dei codici con il proprio token, in modo da poterli eventualmente avviare o poter controllare i risultati. Questa funzione, però, può essere utile più che altro in un utilizzo meno sperimentale di quello che faremo le prime volte che proviamo il calcolatore quantistico.

La funzione **test_api_run_experiment** esemplifica l'esecuzione di un codice OpenQASM sul computer quantistico. Oltre a costruirsi un oggetto per accedere alle API, si deve inserire tutto il codice OpenQASM all'interno di una variabile. Potremmo leggere il codice da un file di testo, ma ovviamente per un semplice test conviene scrivere tutto il codice direttamente nel programma. Ovviamente tutto il codice QASM può essere scritto su una sola riga perché le varie righe possono essere separate con un punto virgola (come nel linguaggio C).

Una opzione da definire è il **device**: può essere di due tipi,

simulator oppure **real**, a seconda del fatto che il codice debba essere eseguito su un simulatore oppure sul vero calcolatore quantistico.

Il parametro **shot** indica il numero di volto in cui si deve eseguire l'esperimento: di solito sul simulatore si fanno 100 cicli, mentre sul computer quantistico almeno 1000. Il massimo è 8192 esecuzioni del codice, ma si può anche provare ad eseguire una sola volta il codice tanto per vedere se funziona.

Infine, si può semplicemente avviare l'esperimento con il metodo **run_experiment** delle API. Il risultato viene fornito sotto forma di array, e ciò che ci interessa è l'elemento **status**.

Di fatto, quindi, eseguire un esperimento è molto semplice, bastano poche righe di codice con le quali si controllano tutti i parametri dell'esperimento e si può leggere il risultato. Python è quindi una ottima opzione per eseguire esperimenti in modo automatizzato. Se si è alle prime armi, tuttavia, conviene utilizzare l'interfaccia web del sito ufficiale per capire come muoversi nella scrittura del codice OpenQASM.

Programmare il calcolatore quantistico

Per imparare il linguaggio nativo del calcolatore quantistico di IBM si può leggere la guida ufficiale, pubblicata assieme ad alcuni esempi su GitHub. Ovviamente è in lingua inglese, ma si occupa di spiegare in modo schematico tutto quello che serve, se non per scrivere dei programmi, almeno per capire

gli esempi. In particolare, a pagina 9 è presente una tabella con i principali comandi del linguaggio. Uno dei metodi migliori per utilizzare davvero il calcolatore quantistico di IBM è sfruttare le API per Python. I programmi "quantistici" non si scrivono in Python, si deve sempre usare il codice nativo OpenQASM, ma Python può essere un modo semplice per lanciare i programmi "quantistici" dal nostro computer.

Il Composer sul sito ufficiale permette di sperimentare con le porte logiche

---

Hello, World!

Hello, world!