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.

Table of Contents

- Come funziona il formato PDF?
- Includere Poppler

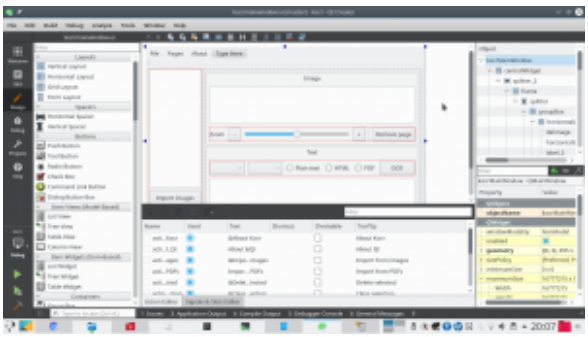
- Aprire un PDF in lettura
- Fusione dei PDF
- Le varie TextBox
- Scrivere dell'HTML
- Il codice sorgente e il binario dell'esempio

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**.

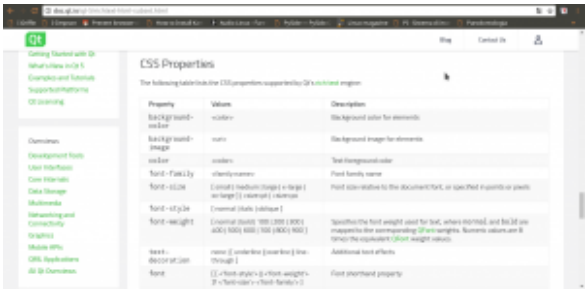
Soltanto dopo avere terminato questo ciclo for, e quindi avere scritto tutti i testi dove necessario, possiamo disegnare sulla pagina l'immagine di sfondo, con la funzione drawPixmap che si usa per inserire in un **painter** una immagine a mappa di pixel (una bitmap qualsiasi). La pixmap è ovviamente 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

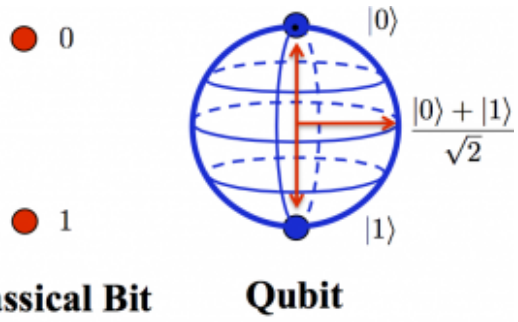
avanti) aumenta diventano molto più difficili e costosi da costruire. Ed è per questo motivo che IBM permette l'accesso a tutti i programmatori tramite il cloud: in questo modo potrà contare su una squadra di alfa-tester volontari, e potrà cercare di migliorare l'efficienza del calcolatore scoprendo 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).

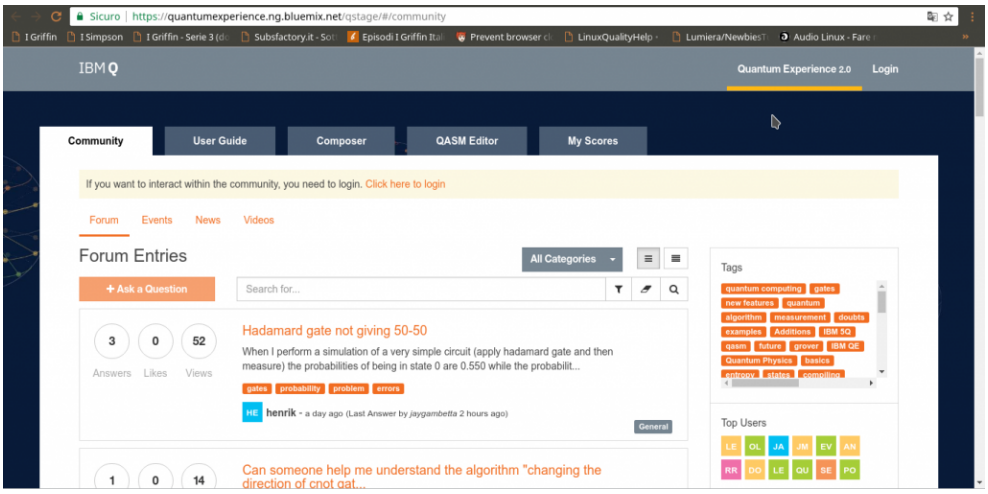
Il quantum computer IBM è gratuito?

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!