SukaSudoku (titolo provvisorio ^__^)

di Davide Eynard

“Immaginate di poter scaricare un decimo dei dati rispetto a quanto siete abituati, evitando tutte le pubblicità e le finestre popup, conservando le informazioni che vi interessano (e solo quelle) sul vostro computer con la possibilità di rivederle quando siete scollegati, in modo personalizzato, più semplice e più efficace. Pensate che, una volta che questi dati sono presenti all’interno del vostro hard disk, potete scrivere dei programmi che operano su di essi per produrre informazioni nuove, ancora più interessanti. Infine, considerate la possibilità di rendere questi dati disponibili a tutti, magari in modo automatico. Ebbene, tutto questo è ciò che io chiamo PowerBrowsing”.

Con questa (auto)citazione, tratta dal sito http://www.powerbrowsing.org, inizia il primo (e, speriamo, non l’ultimo) articolo dedicato al *PowerBrowsing*, un nome sotto al quale si riuniscono diverse tecniche in grado di rendere più efficaci la navigazione, la ricerca e il recupero di informazioni da Internet. Uno dei concetti principali che tali tecniche cercano di trasmettere è che dovremmo sempre avere noi il controllo di ciò che succede all'interno dei nostri computer: il che, tradotto in termini di accesso al Web, consiste nel poter scegliere di vedere solo ciò che desideriamo, nel saper trovare ciò che cerchiamo, e nel saper usare gli strumenti che abbiamo a disposizione (o eventualmente alcuni creati *ad hoc*) per risparmiarci lavoro e fatica.

All’interno di quest’articolo ci concentreremo in particolare su quest’ultimo aspetto, descrivendo lo sviluppo di un robot (o, più semplicemente, *bot*) in grado non solo di navigare all’interno di un sito e scaricare dati per conto nostro, ma anche di riproporli in un formato a noi congeniale. Sull’onda del successo della scorsa estate, e in barba a chi ha voluto sfruttare (se non addirittura creare) quel successo per scopi puramente commerciali, abbiamo deciso di costruire un bot in grado di collezionare dei sudoku dal Web e di pubblicarli all’interno di un documento PDF, già impaginato in forma di libretto e pronto da stampare. Il tutto, naturalmente, utilizzando esclusivamente software libero e sfruttando tecniche che possono essere applicate in numerosi altri casi.

Bot basics

Il termine *robot*, secondo la definizione di Wikipedia (http://www.wikipedia.org), deriva dal termine ceco *robota*, che significa “lavoro pesante” o “lavoro forzato”: molto in generale, quindi, lo scopo di un robot è proprio quello di eseguire al posto nostro compiti di questo tipo. Ora si tratta solo di decidere cosa per noi è considerato “lavoro pesante”: nel campo dell’informatica, infatti, anche le operazioni più complesse non comportano uno sforzo fisico ma, al massimo, sono tremendamente noiose e ripetitive. La navigazione in Internet, in particolare, si riduce il più delle volte a tre semplici operazioni: seguire dei link, esaminare il contenuto delle pagine scaricate ed estrarre nuove informazioni da esse. E questo è proprio il tipo di lavoro che faremo compiere ai nostri bot: raccogliere informazioni per conto nostro.

Ora che abbiamo stabilito le situazioni in cui può tornarci utile *usare* un robot, è bene capire quando invece sia utile *crearne* uno. Se, ad esempio, desiderate scaricare un elenco di file da una pagina o volete salvare sul vostro disco un intero sito, probabilmente non avrete bisogno di costruire un bot: esistono, infatti, diversi programmi già pronti per eseguire questa operazione (`curl` e `wget`, giusto per citarne un paio). Se, invece, desiderate fare qualcosa di particolare per cui ancora non esiste un programma apposito, come ad esempio aggregare informazioni provenienti da diversi siti, o estrarre contenuti da una pagina Web e pubblicarli altrove, allora un bel bot fatto in casa è ciò che fa per voi!

Nella maggior parte dei casi, un robot per il Web dev’essere in grado di eseguire le seguenti operazioni:

- **navigazione**: il robot deve simulare la navigazione di un utente all'interno di un sito Web. Questo significa che esso non deve solamente scaricare dati da Internet, ma anche seguire link in base a schemi prestabiliti, compilare form e così via.
- **estrazione di dati**: il robot dev'essere in grado di filtrare i dati contenuti all'interno di una pagina, eliminando tutto il contenuto inutile (ad esempio, il codice HTML relativo all'impaginazione di un testo) e conservando invece le informazioni considerate interessanti (ad esempio, il testo vero e proprio che viene visualizzato quando si carica una pagina Web).
- **elaborazione e pubblicazione**: le informazioni estratte dal robot devono essere, nel caso più semplice, visualizzate a schermo o salvate su disco. Considerando scenari di utilizzo via via più complessi, potrebbe essere necessario salvare i dati scaricati all'interno di un database, calcolare delle statistiche su di essi o, come nel nostro caso, ripubblicarli un un formato differente.

Più avanti queste operazioni verranno descritte in dettaglio, tramite esempi e programmi scritti in Perl. Abbiamo scelto di utilizzare questo linguaggio di programmazione per diversi motivi: innanzitutto, esso è particolarmente potente e versatile per quanto riguarda l'analisi dei testi; inoltre è multipiattaforma e ha tantissimi moduli già pronti che consentono di eseguire operazioni complesse con estrema semplicità; infine ha una vasta letteratura a proposito, sia in formato elettronico sia cartaceo (come, ad esempio, l'ottimo libro "Spidering Hacks" pubblicato da O'Reilly).

Per lo sviluppo del bot abbiamo deciso di appoggiarci, quando possibile, a software già esistenti, scegliendo applicazioni già presenti all'interno delle principali distribuzioni Linux o comunque facilmente reperibili. Oltre a Perl come linguaggio di programmazione, abbiamo usato LaTeX per la creazione del documento e le psutils (in particolare i programmi psbook, psnup e ps2pdf) per l'impaginazione in forma di libretto. Per il corretto funzionamento del bot, quindi, è necessario che i programmi descritti siano presenti all'interno del vostro sistema: per fortuna, nella maggior parte dei casi è sufficiente installare, tramite il sistema di pacchettizzazione della vostra distribuzione, i pacchetti tetex-bin e psutils e soddisfare automaticamente le loro dipendenze.

Navigazione

Perl dispone di diverse librerie per automatizzare la navigazione del nostro robot: fra queste, forse la più famosa è LWP (libwww-perl), in grado di gestire praticamente ogni aspetto dei collegamenti al Web. Utilizzandola nella sua versione più semplice, è sufficiente un solo comando per scaricare una qualsiasi pagina da Internet:

```
#!/usr/bin/perl
use LWP::Simple;

$contenuto = get ("http://www.google.com");
print $contenuto;
```

All'interno di questo piccolo script Perl, la prima riga rappresenta la classica intestazione, che compare all'inizio di ogni programma, mentre la seconda serve a importare la libreria LWP::Simple, necessaria per poter utilizzare il comando `get`. Attraverso i due comandi successivi, il codice sorgente della pagina presente all'indirizzo http://www.google.com viene prima salvato all'interno della variabile `$contenuto` e quindi stampato a schermo. Se desiderate eseguire questo programma salvatelo all'interno di un file di testo (ad esempio, `prova.pl`) ed eseguitelo con il comando `perl` *`nomefile`* (nel nostro caso, `perl prova.pl`).

Per verificare la semplicità e allo stesso tempo la potenza del comando `get` sono sufficienti alcuni esperimenti: modificando l'URL da scaricare e aggiungendo poche righe di codice è possibile, ad esempio, verificare se una pagina è cambiata, cercare una particolare stringa di testo al suo interno

oppure estrarre alcuni contenuti da essa. Allo stesso tempo, però, vi scontrerete probabilmente con alcuni limiti di tale comando: è sufficiente, infatti, provare a collegarsi con lo stesso script al sito di notizie di Google (http://news.google.it) per ottenere un documento vuoto. Il motivo è che il nostro semplice robot non è in grado di identificarsi con il server e l'accesso al sito di notizie è precluso ai client anonimi.

Naturalmente è possibile risolvere questo problema utilizzando strumenti solo un po' più complessi, ma decisamente più avanzati: ad esempio LWP::UserAgent, una libreria che consente di gestire in dettaglio ogni particolare delle connessioni HTTP. Tramite l'oggetto UserAgent, infatti, è possibile non solo autenticarsi, ma anche ad esempio utilizzare i cookie, definire gli header delle nostre richieste HTTP o selezionare il metodo (GET, POST) con cui vengono inviati i dati al server (per approfondimenti, consultate la pagina http://search.cpan.org/~gaas/libwww-perl-5.805/lib/LWP/UserAgent.pm). Usando questa libreria possiamo creare uno script più evoluto, dotato di controllo degli errori e, già che ci siamo, una piccola funzione di ricerca all'interno della pagina scaricata:

```
#!/usr/bin/perl
use LWP::UserAgent; #usa la nuova libreria

my $ua = LWP::UserAgent->new; #crea un nuovo UserAgent
   $ua->agent('NewsBot/0.1'); #identificazione

my $res = $ua->get("http://news.google.it"); #connessione all'URL

if ($res->is_success){
  if ($res->content =~ /berlusconi/si){
    print "Anche oggi si parla di Berlusconi!\n";
  }else{
    print "This is a Berlusca-free day!\n";
  }
}else{
  #si e' verificato un errore
  die $res->status_line;
}
```

Dopo l'intestazione e l'importazione delle librerie, viene creato un nuovo UserAgent che si identificherà tramite la stringa "NewsBot/0.1". Il nostro UA si connette quindi al sito di news di Google e salva la risposta del server all'interno della variabile `$res`. Tale variabile rappresenta un oggetto di tipo HTTP::Response (descritto anch'esso in dettaglio nella documentazione online), in grado di conservare al suo interno non solo il contenuto della pagina Web richiesta, ma anche l'esito della richiesta stessa, gli header HTTP della risposta ed eventuali messaggi di errore. Se il download ha avuto successo, allora viene eseguita una ricerca all'interno del codice sorgente della pagina scaricata; in caso contrario il programma termina mostrando l'errore generato dal server.

La parte di ricerca all'interno dello script, per quanto semplice, merita un approfondimento. La riga di codice

```
if ($res->content =~ /berlusconi/i){
```

pone una condizione: se, all'interno del codice sorgente della pagina scaricata, viene individuata la stringa "berlusconi", allora viene eseguita una particolare azione, in caso contrario ("else") ne viene eseguita un'altra. Tale condizione viene posta usando una semplice *regular expression*: i due slash (`/`) marcano la stringa da ricercare all'interno del testo, mentre il carattere `i` in fondo serve a specificare che il controllo dev'essere *case insensitive*, cioè non deve tenere conto della differenza fra maiuscole e minuscole.

A questo punto dovreste essere in grado di capire il funzionamento dello script. Naturalmente esso è

ancora molto semplice, sia per quanto riguarda la navigazione sia per il tipo di regular expression utilizzata per la ricerca: mentre quest'ultimo aspetto verrà approfondito solo in seguito, dedichiamoci ora a un'analisi più accurata dei diversi aspetti della navigazione.

Come già accennato in precedenza, ciò che viene definito navigazione è in realtà un insieme delle diverse operazioni che noi eseguiamo ogni volta che ci colleghiamo a un sito Web. Queste comprendono, oltre al semplice download di un URL, la possibilità di seguire uno o più link presenti all'interno della pagina scaricata, oppure la compilazione di un form e l'invio dei suoi dati al server, o ancora il salvataggio di file su disco. Istruire un robot affinché sia in grado di navigare all'interno di un sito significa, innanzitutto, capire quali di queste operazioni esso dovrà compiere e in quale ordine: per questo motivo la prima cosa da fare è un'esplorazione accurata del sito, volta ad individuare particolari schemi di navigazione e a raccogliere informazioni relative alle tecnologie utilizzate.

Come esempio, esploriamo il sito contenente la collezione di sudoku che desideriamo scaricare: il suo indirizzo è http://www.dailysudoku.com ed è stato preso in considerazione non solo per la grande quantità di materiale che mette liberamente a disposizione, ma anche per la grande varietà di operazioni che consente di eseguire. Nella home page vengono pubblicati i sudoku del giorno, divisi in diverse categorie: *Classic*, *Monster*, *Kids* e, eccezionalmente per il periodo Natalizio, *Xmas*; nella sezione *Archive* è possibile trovare tutti i sudoku pubblicati da gennaio 2005 ad oggi; ognuno di essi, infine, è disponibile sia sotto forma di immagine sia come file PDF pronto da stampare.

Chi lo desiderasse, potrebbe scaricare manualmente tutti i sudoku che desidera e stamparli uno per uno all'interno di pagine separate. Noi, tuttavia, abbiamo in mente un progetto più ambizioso: far collegare il robot all'interno della sezione *Archive* e fargli scaricare automaticamente i PDF, per poi reimpaginarli all'interno di un nuovo documento. Per semplicità abbiamo deciso di lavorare solo sui sudoku della categoria *Classic*, ma con poche modifiche potrete far sì che il vostro robot scarichi e impagini correttamente tutto ciò che trova. Dopo una prima esplorazione, i passi individuati per scaricare il PDF di un generico sudoku sono i seguenti:

1. innanzitutto, è necessario aprire la pagina degli archivi, seguendo il link nella home page oppure collegandosi direttamente all'url http://www.dailysudoku.com/cgi-bin/sudoku/archive.pl;
2. all'interno della pagina degli archivi, bisogna scegliere solo i link che portano a sudoku di tipo Classic: da una rapida analisi dei collegamenti, quelli che ci interessano sono solo quelli che contengono all'interno dell'URL la stringa "/sudoku/archive" (potete verificarlo anche voi, passando con il puntatore del mouse sopra ai link e leggendo all'interno della barra di stato dove essi puntano);
3. seguendo uno dei collegamenti selezionati veniamo portati alla pagina del relativo sudoku: per poter estrarre correttamente i dati dobbiamo scaricare il documento in formato PDF, quindi è necessario fare clic sul pulsante "Show page";
4. a questo punto, compare una pagina contenente un redirect: questo significa che nel giro di pochi secondi il browser caricherà automaticamente un altro URL, tuttavia il nostro robot non lo farà a meno che noi non gli ordiniamo di farlo. Per questo motivo, dovremo dirgli esplicitamente di seguire il link specificato all'interno della pagina, associato alle parole "click here".

Durante la fase di esplorazione, alcuni potrebbero aver notato una certa regolarità negli URL dei file PDF: essi, infatti, seguono un formato predefinito e, partendo da una particolare data, è in teoria possibile ricostruire l'indirizzo dal quale scaricare il sudoku. Con queste premesse, replicare i passi di navigazione potrebbe sembrare un'operazione superflua, che rende il robot meno efficiente (deve scaricare più dati per ottenere lo stesso quantitativo di informazioni) e più esposto alla possibilità di errori (le pagine Web cambiano: più sono le pagine da attraversare, più probabilità ci sono che il

robot smetta di funzionare). In linea di massima, queste considerazioni sono corrette ed è solitamente preferibile lavorare, quando possibile, direttamente sugli URL dei documenti a cui siamo interessati. Tuttavia, nella maggior parte dei casi non è possibile generare i link in modo così semplice e la ricostruzione fedele degli URL può diventare un'operazione complicata, se non addirittura impossibile. Inoltre, la "via più lunga" ha in questo caso anche una valenza didattica: attraverso un percorso più intricato, abbiamo la possibilità di spiegare in dettaglio i comandi necessari per portare a termine una maggior varietà di operazioni.

Aumentando la complessità del problema, abbiamo deciso di semplificare gli strumenti per risolverlo: dopo aver descritto LWP::UserAgent, infatti, passiamo a utilizzare una nuova libreria, chiamata WWW::Mechanize. Perché aggiungere un nuovo modulo, quando quello che abbiamo è già in grado di eseguire così tante operazioni? La risposta è semplice: WWW::Mechanize è costruito sopra a LWP::UserAgent (nel gergo della programmazione a oggetti, si dice che il primo è una *sottoclasse* del secondo), e quindi è in grado di fare le stesse cose... Però ne può fare anche altre, e in modo più semplice!

Poiché, a differenza di LWP::UserAgent, WWW::Mechanize non è una libreria solitamente installata insieme all'interprete Perl, è necessario verificare la sua presenza all'interno del sistema ed eventualmente installarla. Impartite da shell il comando

```
perl -MWWW::Mechanize -e exit
```

Se il sistema risponde con un errore (del tipo "Can't locate WWW/Mechanize.pm in @INC"), allora è necessario installare la libreria. In questo caso, aprite la shell di CPAN

```
perl -MCPAN -e shell
```

e impartite il comando

```
install WWW::Mechanize
```

Il programma è in grado di eseguire una verifica delle dipendenze e di scaricare e installare tutti i componenti necessari per il funzionamento di WWW::Mechanize. Al termine dell'installazione, uscite dalla shell di CPAN con il comando exit: a questo punto, potete usare WWW::Mechanize per creare un robot più avanzato, in grado di collegarsi al sito Daily Sudoku e scaricare i PDF seguendo i passi descritti in precedenza. Lo script che segue salva i sudoku nella directory corrente all'interno di file diversi, utilizzando lo stesso nome che essi hanno sul sito:

```
#!/usr/bin/perl

use WWW::Mechanize;

# $URL contiene l'URL della pagina degli archivi.
$URL   = 'http://www.dailysudoku.com/cgi-bin/sudoku/archive.pl';
$SLEEP = 2; # dormi 2 sec. fra una get e l'altra

# crea un nuovo mech
my $mech = WWW::Mechanize->new();

# scarica la pagina degli archivi
$mech->get($URL);

# Per ogni link a sudoku di tipo "Classic", esegui
# le operazioni specificate.
# NOTA : il comando reverse serve a invertire l'ordine
#        dei link (in modo che siano ordinati per data)
# NOTA2: di volta in volta, a ogni iterazione del ciclo
```

```
#        for, l'URL viene salvato nella variabile "$_"

for (reverse $mech->find_all_links(url_regex => qr|/sudoku/archive/|)){
  # dormi $SLEEP secondi
  sleep $SLEEP;
  # converti l'url in formato assoluto
  # e salvalo nella variabile $url_sudo
  my $url_sudo = $_->url_abs;
  # scarica la pagina del sudoku
  $mech->get($url_sudo);
  # fai clic su "Show page" (pulsante del primo form)
  $mech->submit_form;
  # segui il link "click here"
  $mech->follow_link(text=> "click here");
  # estrai il nome del file pdf dall'url
  $mech->uri() =~ m|([^/]*)$|;
  # salva il pdf
  $mech->save_content($1);
  print "Scaricato ".$mech->uri."\n";
}
exit;
```

Nonostante l'esempio sia decisamente più complesso rispetto ai precedenti, noterete che il codice rimane comunque abbastanza comprensibile e, grazie ai commenti, quasi autoesplicativo. Dopo i consueti comandi iniziali, viene creato un nuovo oggetto $mech, istanza della classe WWW::Mechanize. Esso è in grado di eseguire diverse operazioni e, in particolare, è stato utilizzato all'interno dello script per i seguenti scopi:

- scaricare la pagina presente a un determinato URL (`$mech->get($URL)`);
- estrarre, dalla pagina degli archivi, tutti i link che contengono al loro interno la stringa "/sudoku/archive" (`$mech->find_all_links(url_regex => qr|/sudoku/archive/|)`);
- fare il submit di un form (`$mech->submit_form`);
- seguire un link (`$mech->follow_link(text=> "click here")`);
- salvare un file su disco (`$mech->save_content($1)`).

Naturalmente, il robot può essere ancora migliorato: ad esempio, è non solo possibile ma auspicabile l'aggiunta di una serie di controlli sugli errori, in modo da capire se il server non è raggiungibile o se le pagine Web sono state modificate. Inoltre, manca ancora tutta la parte di estrazione dati: lo script, infatti, si limita a salvare i file PDF così come gli arrivano dal server, mentre noi desideriamo prendere da essi solo le informazioni che ci interessano (in particolare, i singoli sudoku) e replicarle nel formato che preferiamo.

Estrazione dati

L'estrazione delle informazioni dai documenti scaricati viene effettuata utilizzando le regular expression di Perl. Questo strumento si rivela particolarmente efficace per l'analisi dei testi, tuttavia ha come contro il fatto di essere piuttosto complesso e decisamente poco leggibile. Tramite questo articolo non pretendiamo di trasformarvi in maghi delle regular expression, ma semplicemente di farvi comprendere il funzionamento di quelle che abbiamo usato, rimandandovi per eventuali approfondimenti all'indirizzo http://www.perl.com/doc/manual/html/pod/perlre.html (o, per chi preferisse dei testi in italiano, gli articoli sulle regexp presenti su http://www.perl.it/documenti/articoli/index.html).

Considerate, ad esempio, le due regexp utilizzate nell'ultimo esempio: mentre la prima (`|/sudoku/archive/|`) è abbastanza semplice da capire, in quanto non fa altro che verificare all'interno di un testo la presenza della stringa specificata; la seconda (`|([^/]*)$|`), invece, è decisamente più criptica. Per comprenderla meglio, spezziamola nei suoi diversi componenti:

- le due barre verticali (`|`) marcano l'inizio e la fine della regexp
- le parentesi quadre stanno a indicare una classe di caratteri e il loro contenuto definisce tale classe per esclusione: i caratteri `^/`, infatti, stanno a significare "qualsiasi carattere tranne lo slash";
- l'asterisco che segue la definizione della classe specifica può presentarsi un numero qualsiasi di caratteri consecutivi (anche zero) appartenenti a tale classe;
- il dollaro segna la fine della riga (in questo caso anche della stringa, trattandosi di un URL);
- le parentesi tonde specificano che, se la stringa in esame soddisfa le condizioni poste dalla regular expression, tutto ciò che è compreso fra di esse verrà salvato all'interno di una variabile riservata. All'interno di ogni regexp possono essere inserite diverse parentesi, e le variabili riservate assumono i nomi $1, $2 e così via, in base all'ordine con cui sono specificate.

Tramite l'uso delle parentesi tonde, le regular expression possono essere applicate a un testo non solo per verificare la presenza di una particolare stringa al suo interno, ma anche per *estrarre* dati da esso. In questo caso siamo riusciti a ricavare il nome del file dato il suo URL completo, mentre il nostro robot dovrà essere in grado di estrarre i sudoku dai relativi file PDF. In realtà, in questo caso siamo stati particolarmente fortunati: non sempre, infatti, i documenti in formato PDF possono essere analizzati come del semplice testo, ma spesso contengono dati in formato binario dai quali non sarebbe stato semplice ricavare informazioni. I file PDF del sito Daily Sudoku, invece, non solo contengono esclusivamente testo, ma presentano anche una sintassi abbastanza facile da comprendere.

Come nel caso della navigazione, infatti, anche per l'estrazione dei dati è necessaria innanzitutto una fase esplorativa, durante la quale il programmatore analizza il formato dei documenti scaricati dal Web, valuta la possibilità di ricavare delle informazioni utili da essi e studia un metodo per estrarre ciò che gli interessa in modo automatico. Per il nostro particolare progetto, desideriamo che il robot recuperi dal PDF i dati relativi al sudoku, cioè i singoli valori (numeri o spazi vuoti) da inserire nelle caselle per replicare lo stesso sudoku nel formato che noi preferiamo. Le conclusioni a cui siamo giunti dopo l'analisi dei file PDF (conclusioni che voi stessi potete verificare, aprendoli con un qualsiasi editor di testo) sono le seguenti:

- i file utilizzano, almeno per ora, un formato comune. In quelli più recenti sono state riscontrate alcune differenze, tuttavia non tali da pregiudicare l'estrazione dei dati;
- le caselle vuote non vengono esplicitamente specificate, mentre quelle contenenti un numero sono codificate tramite una stringa nel formato "`BT /F2 20 Tf y x Td (valore) Tj ET`", dove `x` e `y` segnano le coordinate della casella in cui inserire il valore specificato;
- le coordinate seguono un particolare formato, come mostrato nella Figura 1. Con un paio di semplici formule matematiche, tuttavia, è possibile convertirle in modo che assumano i valori da 0 a 8, più comodi da utilizzare come indici della matrice all'interno della quale vengono salvati i numeri del sudoku.

Il codice per l'estrazione che risulta da queste considerazioni è quello che segue:

```
if ($pag_pdf =~ /435 Td \((.*?)\) .*?435 Td \((.*?)\)/si){
  $caption = "$2 ($1)";
}

# estrai i numeri del sudoku dal pdf
while ($pag_pdf =~ /20 Tf ([^\s]+) ([^\s]+) Td \((\d)\)/gi){
  my $y = ($1 - 189.45) / 30;
  my $x = 8 - (($2 - 458) / 30);
  my $n = $3;
  $sudoku[$x][$y] = "$n";
}
```

All'interno del codice sono presenti due regular expression. La prima viene usata per creare una didascalia del sudoku con data di pubblicazione e difficoltà:

- `435 Td` è la stringa che si trova, all'interno del documento PDF, subito prima del livello di difficoltà e della data;
- `\((.*?)\)` presenta due livelli annidati di parentesi: quelle all'interno sono usate, come descritto in precedenza, per estrarre e salvare in una variabile riservata tutto ciò che si trova al loro interno; quelle all'esterno sono precedute dal simbolo di backslash (\) e rappresentano delle vere e proprie parentesi. In sostanza, questa parte di codice viene letta come "estrai tutto ciò che sta fra parentesi";
- i comandi descritti si ripetono due volte all'interno della regular expression: il risultato è che, se il controllo ha successo, il livello di difficoltà del sudoku verrà salvato nella variabile $1 e la sua data in $2. L'istruzione `$caption = "$2 ($1)";` compone la didascalia unendo i due valori appena estratti.

La seconda regexp viene usata per salvare i valori delle celle del sudoku all'interno di una matrice. Notate che, poiché l'estrazione viene ripetuta su più righe, la regular expression presenta il modificatore `g` (*global*, usato proprio per ripetere la scansione) e il codice, anziché avere una semplice condizione, presenta un ciclo `while`. L'estrazione avviene come segue:

- i gruppi di parentesi, e di conseguenza le variabili riservate all'interno delle quali verranno salvati i valori estratti, sono tre: il primo viene usato per la coordinata y, il secondo per la x e il terzo per il valore da associare ad esse;
- tutto ciò che è fuori dalle parentesi fa parte del normale testo che circonda i valori che desideriamo estrarre;
- i tre gruppi di parentesi contengono, rispettivamente, `[^\s]+` (una stringa di lunghezza maggiore o uguale a uno, che non contiene alcuno spazio) per le coordinate e `\d` (una singola cifra) per il numero da inserire nel sudoku.

I comandi contenuti all'interno del ciclo while, infine, convertono i valori delle coordinate x ed y in numeri compresi fra 0 e 8, che verranno poi usati come indici della matrice `$sudoku`. All'uscita dal ciclo avremo quindi una replica del sudoku originale, pronta per essere usata come desideriamo.

Elaborazione e pubblicazione

L'opera di pubblicazione è, fra quelle descritte finora, forse la più semplice. Infatti, grazie a diversi

contributi esterni sotto forma di applicazioni e librerie, i comandi da eseguire in Perl per pubblicare in forma di libro i sudoku estratti dal nostro robot si limitano al salvataggio dei dati, in formato testo, all'interno di un modello già pronto. Il resto delle operazioni (compilazione in formato Postscript, creazione del libro, riduzione delle pagine) si riduce a una serie di comandi, da salvare all'interno di uno script o da eseguire manualmente via linea di comando.

Una volta estratti i dati lo script Perl li converte in formato testo, pronti per essere inclusi all'interno di un documento LaTeX. Esso apre quindi un template come quello mostrato nel Riquadro 2 e lo completa con i dati appena convertiti: la sostituzione `s/###SUDOKI###/$sudoki/`, infatti, serve proprio a questo scopo. Infine salva il tutto all'interno di un file (nello script, il suo nome è `sudoki.tex`). Il codice dello script Perl completo si trova all'interno del Riquadro 1.

Poiché il template che abbiamo utilizzato fa uso di una libreria appositamente creata per stampare i sudoku, è necessario che essa sia presente al momento della compilazione del file .tex. Il modo più veloce per ottenerla è il seguente:

1. collegarsi al sito http://tug.ctan.org/tex-archive/macros/latex/contrib/sudoku/ e scaricare il file `sudoku.zip`;
2. estrarre i contenuti del file, accedere alla directory `sudoku` ed eseguire il comando `latex sudoku.ins`
3. copiare il file `sudoku.sty` all'interno della stessa directory in cui è salvato il file `sudoki.tex`.

Una volta copiata la libreria, è finalmente possibile creare il libro dei sudoku: innanzitutto, è necessario eseguire `latex` per compilare il sorgente in un file DVI; con il comando `dvips` si converte il file in formato Postscript, pronto per essere impaginato con `psbook`; `psnup` riduce la dimensione delle pagine, in modo da creare un libretto grande la metà di un foglio A4; infine `ps2pdf` converte il file Postscript in un documento PDF, pronto da stampare o da inviare agli amici. La sequenza delle operazioni (seguita da una serie di comandi per ripulire la directory corrente da file temporanei) è mostrata nello script all'interno del Riquadro 3.

Conclusioni

Quando si parla di robot e agenti per il recupero automatico di informazioni da Internet, un aspetto che non bisogna mai tralasciare è quello della netiquette. Se, infatti, è vero che anche quando navighiamo normalmente ci sono alcune regole di comportamento che sarebbe bene rispettare, questo diventa ancora più importante quando a fare le nostre veci è un programma, molto più veloce di noi e in grado di lavorare senza arrestarsi mai.

Una delle cose a cui è bene pensare in questi casi è che le risorse dei server a cui ci colleghiamo non sono sempre infinite: ognuno di essi ha una banda che, per quanto grande, è comunque limitata, e magari un abbonamento il cui prezzo varia in base alla quantità di dati trasferiti. A maggior ragione, quando un sito non commerciale offre gratuitamente un servizio a tutti quanti è bene non abusare di esso. Questo è uno dei principali motivi per cui lo script creato non scarica tutti i file presenti in archivio, ma solo quelli del mese corrente, e interpone una breve pausa fra un download e l'altro in modo da non intasare di richieste il server. Naturalmente nessuno vi impedisce di modificare il codice Perl come meglio credete, togliendo limitazioni e ampliandone il raggio d'azione, ma il nostro consiglio è quello di utilizzarlo principalmente a scopo didattico: con quello che avete imparato all'interno di questo articolo, infatti, siete già in grado di scrivere nuovi robot per soddisfare le vostre esigenze.

Infine, se i sudoku non vi bastano mai, tenete presente che non siete necessariamente obbligati a scaricarli da Internet: il programma gnome-sudoku (http://gnome-sudoku.sourceforge.net), ad

esempio, è in grado di generare sudoku sempre nuovi e vi permette di risolverli direttamente dal vostro computer (v. Figura 2). Su Doku (http://sudoku.sourceforge.net), inoltre, permette di salvarli all'interno di un file di testo: per la gioia degli amanti del sudoku in formato cartaceo, con un piccolo hack (v. Riquadro 4) potete modificare lo script Perl in modo che recuperi i sudoku dal file... E creare tutti i libri che desiderate!

Riquadro 1: il codice Perl del robot nella sua versione più completa

```
#!/usr/bin/perl
#
use WWW::Mechanize;

$URL     = 'http://www.dailysudoku.co.uk/cgi-bin/sudoku/archive.pl';
$TPLFILE = 'tpl_sudoku.tex'; # nome del template per generare LaTeX
$OUTFILE = 'sudoki.tex';     # nome del file LaTeX generato
$SLEEP   = 2;                # dormi 2 sec. fra una get e l'altra
$|       = 1;                # non bufferizzare l'output

# crea un nuovo mech
my $mech = WWW::Mechanize->new();

# scarica la pagina degli archivi
my $res = $mech->get($URL);

# lavora sui link
for (reverse $mech->find_all_links(url_regex => qr|/sudoku/archive/|)){
  sleep $SLEEP;
  my $url_sudo = $_->url_abs;
  print "Working on $url_sudo:\n";

  print "\tDownloading url... ";
  $res = $mech->get($url_sudo);
  die "Error downloading url\n" unless $res->is_success;

  print "Submitting form... ";
  $res = $mech->submit_form;
  die "Error submitting form\n" unless $res->is_success;

  print "Downloading pdf... ";
  $res = $mech->follow_link(text=> "click here");
  die "Error downloading pdf\n" unless (defined($res) && $res->is_success);

  # in $pag_pdf viene salvato il pdf del sudoku
  my $pag_pdf = $res->content;

  my @sudoku;
  my $caption;

  # estrai la didascalia dal pdf
  if ($pag_pdf =~ /435 Td \((.*?)\) .*?435 Td \((.*?)\)/si){
    $caption = "$2 ($1)";
  }

  # estrai i numeri del sudoku dal pdf
  while ($pag_pdf =~ /20 Tf ([^\s]+) ([^\s]+) Td \((\d)\)/gi){
    my $y = ($1 - 189.45) / 30;
    my $x = 8 - (($2 - 458) / 30);
    my $n = $3;
    $sudoku[$x][$y] = "$n";
  }

  my $sudo = toLatex(toText(@sudoku),$caption);
  push @sudokistack, $sudo;

  print "Ok\n";
}

# crea la stringa contenente tutti i sudoki
```

```
for (@sudokistack){
  $sudoki .= $_;
}

# carica il file template
my $TPL = "";
open (IN, "<$TPLFILE");
while (<IN>){
  $TPL .= $_;
}
close IN;

$TPL =~ s/###SUDOKI###/$sudoki/;

open (OUT, ">$OUTFILE");
print OUT $TPL;
close OUT;


exit;

# ------------------------------------------------------------------------
# la funzione toText riceve in ingresso un array contenente il sudoku e
# restituisce una stringa che lo rappresenta
#
sub toText {
  # @sudoku e' l'array 9x9 contenente il sudoku
  my @sudoku = @_;
  # $sudo e' la stringa che conterra' il sudoku convertito in testo
  my $sudo = "";

  for ($i=0;$i<9;$i++){
    $sudo .= "|"; #inizio riga
    for ($j=0;$j<9;$j++){
      $sudo .= $sudoku[$i][$j]."|"; #casella sudoku
    }
    $sudo .= ".\n"; #fine riga
  }
  return $sudo;
}

# ------------------------------------------------------------------------
# la funzione toLatex riceve un sudoku in formato stringa (in particolare
# nel formato restituito da toText) e una didascalia da inserire sotto il
# sudoku. Restituisce del testo LaTeX, contenente il codice utile per la
# corretta visualizzazione del sudoku.
sub toLatex {
  my ($sudo,$caption) = @_;
  my $tex = "";
  $tex .= "\\begin{sudoku}\n$sudo\\end{sudoku}\n"; #sudoku
  $tex .= "\\begin{center}\n$caption\n\\end{center}\n"; #didascalia
  if ($flag){
    # ogni 2 sudoku inserisci un pagebreak: questo ramo dell'if
    # e il successivo vengono eseguiti alternativamente
    $tex .= "\n\\pagebreak\n\n";
    $flag = 0;
  }else{
    $tex .= "\n\\verb||\n\n";
    $flag = 1;
  }
  return $tex;
}
```

Riquadro 2: il template LaTeX usato per pubblicare i sudoku

```
\documentclass[12pt,a4paper]{report}
\usepackage{sudoku}

\setlength{\voffset}{0pt}
\setlength{\topmargin}{0pt}
\setlength{\headheight}{0pt}
\setlength{\headsep}{0pt}
\setlength{\textheight}{690pt}


\begin{document}

###SUDOKI###

\end{document}
```

Riquadro 3: lo script usato per creare il libro a partire dal codice LaTeX

```
#!/bin/sh
latex $1
dvips $1
psbook $1.ps tmp.ps
psnup -2 tmp.ps >tmp2.ps
ps2pdf tmp2.ps $1.pdf
rm tmp.ps
rm tmp2.ps
rm $1.ps
rm $1.dvi
rm *.aux
rm *.log
```

Riquadro 4: un'alternativa "locale" al robot, anch'essa in grado di generare un libretto in pdf

Se non vi sono sufficienti i sudoku che trovate online, potete generarne sempre di nuovi con il programma SuDoku (http://sudoku.sourceforge.net), salvarli in un file di testo e impaginarli con lo script Perl modificato come segue. Scaricate il binario Java (necessita della JRE 5.0 per funzionare) dal sito e usate il seguente comando per generare 100 nuovi sudoku:

```
java -cp sudoku_binary_R1_versione.jar com/act365/sudoku/Composer -mu 50 -ms 100
-c 20 24 >sudoku.txt
```

Quindi, modificate l'inizio del vostro script Perl come segue e salvatelo. Supponendo si chiami bot.pl, eseguitelo con il comando:

```
perl bot.pl <sudoku.txt
```

Dopo l'esecuzione troverete su disco un file, chiamato `localsudoki.tex`, pronto da compilare e impaginare.

```
#!/usr/bin/perl
#

$TPLFILE = 'tpl_sudoku.tex'; # nome del template per generare LaTeX
$OUTFILE = 'localsudoki.tex';     # nome del file LaTeX generato

my $content = "";
while (<>){
  $content.=$_;
}


while ($content =~ /Time.*?\n(.+?)\n(Puzzle|\d+ solutions)/gs){

  my @sudoku;
  my $sudoku_text = $1;
  my $i = 0;
  while ($sudoku_text =~ /((\d|\.){1})/gs){
    my $val = $1;
    $val = " " if $val == '.';
    my $x = int($i / 9);
    my $y = $i % 9;

    $sudoku[$x][$y] = $val;
    $i++;
  }
  my $sudo = toLatex(toText(@sudoku),$caption);
  push @sudokistack, $sudo;
}

# crea la stringa contenente tutti i sudoki
for (@sudokistack){
...
```