Il formato testo (e altri formati testuali)

Marco Baroni

12 novembre 2004

1 Il testo dal punto di vista dei computer

1.1 Bits, bytes, caratteri e codici

- Per un sistema operativo, un *file* è una sequenza di *bits* (numeri binari: 0/1).
- Una sequenza di 8 bits costituisce un *byte*.
- Un file di testo è semplicemente un file in cui ciascun byte viene usato per rappresentare lettere dell'alfabeto, numeri, simboli, caratteri di controllo (a capo e simili), seguendo un *codice* (tabella di corrispondenza tra bytes e caratteri) ben definito.
- Un *programma* in grado di manipolare files di testo (per es., `more`), legge la sequenza di bytes contenuti in un file e tratta ciascuna sequenza di 8 bits (ciascun byte) come un carattere.
- Vari codici standardizzati specificano corrispondenza tra bytes e caratteri.
- Uno dei primi e il più famoso è il codice ASCII.
- Il codice più diffuso per le lingue dell'Europa occidentale (incluse lingue "minori" quali basco, faroese e finlandese) è un'estensione dell'ASCII nota come latin-1 (o ISO-8859-1).
- Esempi:

```
01000001 -> A
01100001 -> a
00110010 -> 2
11101000 -> e con accento grave
00001010 -> \n (new line)
```

- Per noi umani (programmatori a parte), i numeri binari non sono il massimo, e dunque le tabelle ASCII/latin-1 tipicamente riportano i numeri corrispondenti in base 10 (decimale), 8 (ottale) o 16 (esadecimale)[1]

[1] A dire il vero, gran parte degli umani non programmatori si trovano bene soltanto con i decimali.

carattere	binario	ottale	esadecimale	decimale
A	01000001	101	41	65
a	01100001	141	61	97
2	00110010	62	32	50
è	11101000	350	E8	232
\n	00001010	12	A	10

Tabella 1: Alcuni caratteri e bytes corrispondenti in latin-1

- Esaminare un file di testo in ottali:

  ```
  $ hexdump -bc brown.txt | more
  ```

- Prima di abbandonare l'affascinante mondo dei bits, si noti che con 8 bits (cioè un byte) possiamo rappresentare 256 oggetti distinti (2^8: da 00000000 a 11111111).

- Dunque, un codice basato su bytes singoli non può rappresentare più di 256 caratteri.

- Conseguenze:

  - Proliferare di standard diversi per lingue diverse (in parte, sotto l'egida dell'ISO).
  - Sistemi a più bytes per lingue con alfabeti molto estesi (cinese, giapponese...)

- Unicode: la soluzione?

  - Codice "standard" che si propone di rappresentare *tutti i caratteri conosciuti*.
  - Definito in astratto, implementazioni differiscono (le più comuni: utf-8, utf-16), molta confusione tra i non esperti.
  - Unicode-support non ancora molto diffuso (lo sarà mai?)
  - The good news: latin-1 è un sotto-insieme di unicode/utf-8.

1.2 Andare a capo con dos, mac e unix

- Caratteri di controllo in ASCII/latin-1 ci riportano al tempo in cui i computer mandavano fisicamente l'output in stampa ad una telescrivente (una specie di macchina da scrivere controllata dal computer).

- Al tempo, quando voleva andare a capo, il computer doveva dare due istruzioni alla telescrivente:

  - Riporta il carrello a sinistra.
  - Scorri in giù di una riga.

- Dunque, servivano due caratteri di controllo: il *carriage return* (CR, \r, decimale 13) e la newline o *line feed* (LF, \n, decimale 10).
- Con l'avvento dei monitor, non c'è più bisogno di dare istruzioni per lo spostamento meccanico di una testina, dunque non è necessario rappresentare l'*a capo* come una sequenza di due caratteri.
- Con grande senso d'armonia, le tre famiglie di sistemi operativi più comuni hanno scelto tre convenzioni diverse:
  - Dos: CR LF
  - Mac: CR
  - Unix: LF
- I problemi che si incontrano trasferendo un file di testo da un sistema all'altro dipendono quasi sempre dal modo diverso di indicare gli *a capo*.
- Sono problemi che si possono risolvere facilmente con un po' di *sed*:

```
$ sed "s/$/\r/" blah.unix > blah.dos
$ sed "s/\r//" blah.dos > blah.unix
```

2 Testo puro vs. altri formati per rappresentare dati testuali

- Il testo puro è il formato più adatto (anzi, l'unico formato proponibile) per la gestione di corpora e altri database linguistici (lessici, liste di frequenza, sistemi di relazioni concettuali...)[2]
- Formati quali *pdf* e *doc* (il formato dei files di MS Word) possono essere più adeguati per altri scopi (per esempio, possono essere più piacevoli all'occhio umano), ma non sono utilizzabili per l'analisi linguistica.
- Il vantaggio del formato testo sta nella sua semplicità: un file di testo è costituito da una sequenza di bytes, dove ciascun byte rappresenta un carattere secondo un codice pubblicamente disponibile.[3]
- La semplicità del formato testo fa sì che praticamente tutti i tools unix e non, gli editors e in generale tutte le applicazioni per la manipolazione di dati testuali, lo supportino.
- Inoltre, è estremamente facile sviluppare nuove applicazioni che gestiscano il formato testo.

[2] Alcune applicazioni linguistiche, come WordSmith Tools e l'IMS Corpus Workbench, mantengono i dati in un loro formato interno. Tuttavia, anche queste applicazioni richiedono testo puro in input, e possono produrre testo puro come output.

[3] Se il file si basa su codici per lingue asiatiche o certe implementazioni di unicode, ciascun carattere è rappresentato da una sequenza di bytes, ma l'idea di base è la stessa.

- Ovviamente, anche i files in altri formati, quali *pdf* e *doc*, dal punto di vista del computer, sono sequenze di bits.
- Tuttavia, solo certi programmi (MS Word, Acrobat Reader) sono in grado di interpretare correttamente tali sequenze di bits.
- Nel caso di *doc* e altri formati della Microsoft, la situazione è aggravata (dal nostro punto di vista) dal fatto che si tratta di formati *proprietari*: ossia, in parole povere, la maniera in cui i bits vanno organizzati per costruire un file *doc* (o *xls*, o *ppt*...) di senso compiuto è un segreto industriale della Microsoft.
- Di conseguenza, dati in questi formati possono venire utilizzati solo con programmi Microsoft, e sono inutili ai nostri fini.
- Ovviamente, un file in formato *pdf* o *doc* può venir letto anche da un programma che si aspetta testo puro.
- Tuttavia, tale programma tratterà ogni sequenza di 8 bits nei files in questione come un carattere ASCII/latin-1, e il risultato non avrà molto senso.
- Provate a leggere i files *Autunno2004_EN.doc* e *collocations.pdf* (nella cartella *cl_shared_data/formati*) con `more`.
- `more` cerca di fare il suo dovere, ma i "caratteri" in cui converte i bytes in questi documenti non hanno molto senso.
- Se proprio bisogna lavorare con formati di questo tipo, esistono command line tools (quali `antiword` e `pdftotext`) che convertono da questi formati a testo, con risultati alterni.[4]

3 HTML e altri markup languages

- L'HTML, il linguaggio nel quale è prodotta la stragrande maggioranza delle pagine web, viene scritto in files di testo puro.
- La formattazione e altri aspetti non strettamente testuali (in particolare: collegamenti intra- ed inter-testuali) non sono espressi al livello dei bits, ma esplicitamente come istruzioni specificate all'interno del testo.
- Per esempio, una parola in grassetto in html si rappresenta così:

```
<b>hi</b>
```

[4] Tra l'altro, i files *doc* salvati come testo seguono un codice che è simile al latin-1, ma non è proprio latin-1, creando una serie di piccoli ma fastidiosi problemi in fase di tokenizzazione.

- Una sequenza di questo genere viene trattata come normalissimo testo da un programma come `more`, mentre programmi speciali (*in primis*, web browsers quali Mozilla/Firefox e Internet Explorer) interpretano le istruzioni di formattazione in HTML, visualizzando la pagina come specificato (in questo caso, nascondendo `<b>` e `</b>` e facendo vedere la stringa **hi** in grassetto).

- Provate a visualizzare la pagina *compling_materials.html* con `more`, e paragonate quello che vedete con la stessa pagina aperta con Internet Explorer.

- Dal nostro punto di vista, l'HTML è un'ottima cosa: significa che gran parte delle pagine web sono già in formato testo – pronte per essere usate come corpora!

- Ci sono programmi più sofisticati per rimuovere le *tags* (`<b>` et sim.) dall'HTML, ma spesso un po' di *sed* può bastare:[5]

```
$ sed "s/<[^>]*>//g" compling_materials.html  > ~/out.txt
```

- Ci riferiamo a linguaggi quali l'HTML, dove informazioni metatestuali di vario genere vengono espresse in testo puro usando una sintassi particolare, con il termine di *markup languages*.

- Un altro markup language famoso è quello usato per l'input al sistema di typesetting LaTeX, con cui scrivo gran parte dei miei documenti (per es., l'input da cui ho generato l'handout sulle collocazioni è nel file *collocations.tex*, che, essendo testo puro, potete tranquillamente leggere con `more`).

- Il markup language più trendy da qualche anno a questa parte è XML, un "meta-linguaggio" adatto a rappresentare dati strutturati.

- XML, tra l'altro, è un ottimo linguaggio per codificare lessici, database terminologici e simili:

```
<lexicon lang=''it''>
<form id=''1'' eng_corresp_id=''33''>
<lemma>cane</lemma>
<pos>N</pos>
<meaning>essere peloso che abbaia</meaning>
</form>
<form id=''2'' eng_corresp_id=''12''>
<lemma>gatto</lemma>
<pos>N</pos>
```

[5] In HTML, i caratteri accentati e vari altri simboli possono venire rappresentati con sequenze speciali chiamate *entity references*. Per esempio, la *ò* è rappresentata da *ò*, la *é* da *é* e via dicendo. In linea di massima, è una buona idea convertire queste sequenze nei caratteri latin-1 corrispondenti prima di procedere all'analisi testuale.

```
<meaning>essere peloso che miagola</meaning>
</form>
</lexicon>
```

- Se il corso di Linguistica Computazionale durasse il doppio del tempo, dedicherei senza dubbio un po' di spazio ad XML.