情報処理 II 第 7 回

情報の電子化 (2) テキスト・ファイルの処理

桂田(かつらだ) 祐史(まさし)

2001 年 6 月 7 日

前回、テキスト・ファイルの構造について説明したが (実に簡単なものだった)、今回はプログラムを使ってテキスト・ファイルに色々な処理を施してみる。実際にキーボードを叩いて体験してもらいたい。

1 UNIX のテキスト・ファイル処理

文書は電子化されてしまえば、プログラムで様々な処理をすることが可能になる。UNIX にはテキスト・ファイルの処理のために便利な仕掛け (入出力のリダイレクション、パイプ) が用意されていて、すぐれたテキスト処理コマンド (ただし基本的に英語向け) が用意されている。その一端を体験して「どういうことが出来るのか」を知ろう。— コマンドの使い方を覚えて欲しいわけではない。

1.1 標準入出力のリダイレクション、パイプ

UNIX において標準入出力のリダイレクション、パイプは重要な機能である。

- UNIX システムでは、各プロセス[1]は、標準入力 (standard input)、標準出力 (standard output) と呼ばれるデータの出入口を持つ。
- 例えば C 言語の `scanf()`, `getchar()` 等は標準入力 (stdin) から入力し、`printf()`, `putchar()` は標準出力 (stdout) に出力する。
- 通常は次のように結びつけられている:
  - 標準入力はキーボードからの入力に
  - 標準出力は画面への出力に

  この結合は簡単に変更できる (標準入出力のリダイレクション と呼ぶ)。
  - コマンド > ファイル名 とすると、コマンドの標準出力を (画面の代わりに) 指定したファイルに書き出す。

[1]動いているプログラムのことをプロセスと呼ぶ。

cal コマンドの出力を cal.out にリダイレクト

```
tango21% cal
      2001年  6月
日 月 火 水 木 金 土
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

tango21% cal > cal.out
tango21% cat cal.out
      2001年  6月
日 月 火 水 木 金 土
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

tango21%
```

- コマンド < ファイル名 とすると、(キーボードの代わりに) 指定したファイルからコマンドの標準入力に読み込む[2]。

- コマンド 1 | コマンド 2 とすると、コマンド 1 の標準出力をコマンド 2 の標準入力に渡すことが出来る。つまり

```
prog1 > datafile
prog2 < datafile
```

のようにファイルを介して prog1 の出力を prog2 に渡す代わりに

```
prog1 | prog2
```

で済ますことが出来る。この機能をパイプと呼ぶ。実例は以下で (山のように) たくさん見せる。

1.2 フィルター

まず以下の準備をしてから、その後の操作をしよう。(tcsh の設定がまだの人は、 tcsh とタイプして、 tcsh を起動してから実習を始めて下さい。)

[2]最近では cat ファイル名 | コマンド とパイプを使ってすますことが多くなって、入力のリダイレクトはあまり使われなくなったような気がする。

```
waltz11% cd                                ← ホームディレクトリィに移動する。
waltz11% mkdir filter                      ← filter という名前のディレクトリィを作る。
waltz11% cd filter                         ← filter に移動する。
waltz11% ln -s ~re00018/gutenberg/* .      ← ~re00018/gutenberg にあるファイルにシンボリック・
                                             リンクを張る。
waltz11% ls                                ← どういうファイルがあるか、確認する。
                                           → alice29.txt などのファイル名が見えるはず。
waltz11% cat LIST                          ← LIST の中身を見る。
```

今回配布したファイル (以下では Gutenberg テキストと呼ぶことにする) の出所については、付録 A. 「古典 (文書) の電子化」を参照してほしい。

UNIX システムでは、標準入出力のリダイレクション、パイプ機能を活用するため、

標準入力から入力し、標準出力に出力し、他の入出力は使わないで済む

というコマンドが多い。このようなプログラムをフィルターと呼ぶ。フィルターは簡単な機能しか持たないものが多いが

複数のフィルターをパイプで接続して複雑な仕事が出来る

ようになっている[3]。

以下、 UNIX の代表的なフィルターをいくつか紹介する。

grep テキスト・ファイルから指定した文字列を検索して、それを含む行を表示する。

検索に用いるパターンの指定には、正規表現 (**regular expression**[4]) と呼ばれる形式が利用できる (grep の名前の由来は global/regular-expression/print)。

- grep　パターン　検索したいファイル名 が基本的な使い方。

grep で指定したパターンを含む行の表示

```
waltz21% grep Alice alice29.txt      alice29.txt から Alice という語を含む行を
                                     探す。
waltz21% grep 'ee[09]80' ee-list     数学科学生のパスワード・マップ ee-list から
                                     ee080, ee980 を含む行を探す。
```

- -v とすると、パターンを含まない行を表示する。

grep -v で指定したパターンを含まない行を表示

```
waltz21% grep -v tcsh ee-list   ee-list から tcsh を含まない行を探す。
```

- -i とすると、大文字・小文字を区別しないで検索する。

grep -i で大文字小文字を区別しない検索

```
waltz21% grep -i 'MARK TWAIN' *   カレント・ディレクトリィから “MARK TWAIN” を含む
                                  ファイルを探す (大・小文字の違いは無視)。
```

[3]それから、これは機能というわけではないが、フィルターは高速に実行できるよう、相当に高度なチューニングがされているのが普通である。

[4]正規表現で記述できる言語を処理するアルゴリズムは簡単で、プログラムも自動的に作成可能である。 UNIX では、文字列パターンを表すのに、正規表現が広く使えるようになっている。

sort テキストファイルを行単位で、順序付けて (通常はアルファベット順に) 並べ変える。

- -r　逆順に並べる。
- -n　数値としての大小で並べ変える。
- +x (x は整数)　第 x フィールドの要素の大小で並べる。

例えば、

ホームディレクトリィにあるファイルを、サイズが大きい順にリストする

```
waltz21% ls -l ~ | sort +4 -n -r
```

(ここでは ls -l ~ の出力の第 5 フィールドはファイルのサイズである、と仮定している。)

wc テキスト・ファイル中の行数、単語数、文字数を数える。

wc で行数、単語数、文字数を数える

```
waltz21% wc alice29.txt
```

uniq 連続する同じ内容の行を 1 つにまとめる。
-c とすると、同じ行が何行続いたか表示する。次の例は、今回のハイライトとも言える凝った例である。

呪文のように見えるかもしれませんが...

```
waltz21% cat alice-words | sort | uniq -c | sort -n
```

一体何をやっているのか? 一段一段確認しながらチェックしよう。
長いコマンドはシェル・スクリプト[5] にすると便利である。

シェルスクリプトの利用

```
waltz21% cat top20
#!/bin/sh
cat "$@" | sort | uniq -c | sort -r -n | head -20
waltz21% ./top20 alice-words
```

/usr/ucb/tr 文字単位の置換、削除を行う。
/usr/ucb/tr [-cds] [文字列 1 [文字列 2]]

- -d は削除することを表す。
- -s は同じ字が連続した場合、一字に置換することを表す。
- -c はパターンに含まれない文字を対象にする。

[5]コマンドをシェルの文法に則って並べて作ったテキスト・ファイルを書き、chmod +x ファイル名 として作成できる実行可能なコマンドのことをシェル・スクリプト (shell script) と呼ぶ。script には台本という意味がある。例に上げたシェル・スクリプト top20 は 2 行 (実質 1 行) のテキスト・ファイルである。

パターンには文字コードを 8 進数で指定できる (以下の例を参照) [6] 。また文字の範囲を - で指定できる。 A-Z で 'A' から 'Z' までの文字 (つまり英大文字) を表す。

以下で /usr/ucb/tr と長く打つのが面倒と思う人は、最初に set path=(/usr/ucb $path) としておくと、以下は単に tr とすることができる。

BSD UNIX の tr (/usr/ucb/tr)

```
waltz21% cat alice29.txt | /usr/ucb/tr a-z A-Z             小文字を大文字にする。
waltz21% cat alice29.txt | /usr/ucb/tr A-Z a-z             大文字を小文字にする。
waltz21% cat DOS-text.txt | /usr/ucb/tr -d "\015\032"      コードが 015, 032 の文字を
                                                           削除する。
waltz21% cat alice29.txt | /usr/ucb/tr -cs A-Za-z '\012'   アルファベット以外の文字を
                                                           改行に変換。
                                                           → 一行一単語に分解する。
```

上記の alice-words というファイルは次のようにして作った (つまりアルファベット以外の文字の連なりを改行に変換した)。

alice-words はこうして作った

```
waltz21% cat alice29.txt | /usr/ucb/tr -cs A-Za-z '\012' > alice-words
```

だから、単語の出現頻度表を得るために、次のようにすることもできる。

```
waltz21% cat alice29.txt | /usr/ucb/tr -cs A-Za-z '\012' | sort | uniq | sort -n
```

(こんなに長いコマンドは打つ気になれない？いざとなったらシェル・スクリプト！)

head 最初の数行のみ表示するコマンド。

オプションを指定しないと最初の 10 行を表示する。 -x とすると最初の x 行を表示する。

```
waltz21% head -30 /usr/dict/words    ← /usr/dict/word の先頭から 30 行を表示する。
```

(/usr/dict/words は英単語を納めたファイルである。)

tail テキストの最後の方を表示するコマンド。

オプションを何も指定しないと 10 行のみ表示する。 -x とすると最後の x 行を表示する。 +x とすると、最初の x 行を除いた残りを表示する。

```
waltz21% cat sawyr10.txt | tail -30    ← sawyr10.txt の最後の 30 行を表示する。
```

nkf 日本語の文字コード (漢字コード) の変換をする。

(nkf については、前回のプリントに最低限のことは説明したので、ここでは略する。)

[6] 015 (8 進) $= 1 \times 8 + 5 = 13$ は C-m (復帰, carriage return) の文字コード、 032 (8 進) $= 3 \times 8 + 2 = 26$ は C-z (古い MS-DOS テキスト・ファイルの終り) の文字コード、 012 (8 進) $= 1 \times 8 + 2 = 10$ は C-j (改行, newline (NL), linefeed (LF) とも) の文字コード。

1.3 awk

awk(オーク) は強力な文字列処理機能と、 C 言語に似た文法を併せ持つ言語処理系であるが、フィルターとしての性格を持ったコマンドなので[7]、ここで紹介しておく。

多分 C 言語を知っている人には、次のプログラムを理解するのは難しくないであろう (別に今理解する必要はない)。先頭に ee080 というパターンを持つ行の第 1, 5, 7 フィールドを表示し、第 7 フィールドが /bin/csh である数を数え、その割合を求めている。

awk スクリプトの例 — test.awk

```
#!/usr/meiji/gnu/bin/gawk -f
BEGIN {
        FS = ":";
        number = 0;
        numofcsh = 0;
}

/^ee080/ {
        printf("%s %20s, %s\n", $1, $5, $7);
        number++;
        if ($7 == "/bin/csh")
              numofcsh++;
}

END {
        printf("%d 人中 csh を使っているのは %d 人 (%4.1f\%) です。\n",
               number, numofcsh, 100.0 * numofcsh / number);
}
```

という awk のプログラム[8] test.awk を使って、

```
./test.awk ee-list
```

とする。
(ちなみに、私はこの gawk を用いて、レポートや試験の採点処理をしている。)

2 レポート課題 4

〆切は 6 月末日。

課題 4 英文中のアルファベットの出現頻度は ‘e’ が一番高く、その次は... などと言われ、古典的な推理小説の暗号の話の種になったりしている。 Gutenberg Project の中のテキストで、そのことを確かめて見よ。手作業ではなく、なるべくコンピューターにやらせること。テキストごとに大きな違いがあるか? 文字が別の記号に置き換えられた場合、出現頻度情報から解読することの可能性について考えよ (要するに他の文字の出現頻度はどの程度まで一定しているのか調べる — 実際に試してみると良いのだけど)。なお、文字の頻度を調べる hindo.c というプログラムを用意した。 (このプログラムは文字の出現頻度順に

[7]似たような性格を持っているコマンドに Perl(パール) がある。 Perl は WWW の CGI のプログラムを記述する言語として普及している。
[8]これも (シェル・スクリプトと同様のニュアンスで) awk スクリプトと呼ばれることが多い。

は表示しないが、 sort を使えば簡単に頻度順に並べられる。どうすればいいか？今回説明した話の簡単な応用である。)

```
waltz21% cc -o hindo hindo.c
waltz21% cat hindo.c | ./hindo
```

テキスト、あるいは作家ごとに単語の使用頻度の癖のようなものがあると思われるが、そのことを Gutenberg テキストで実際に調べてみよ。ルイス・キャロルとマークトウェインの書いたものにどの程度の差があるか？

また、できれば ~re00018/gutenberg/ にあるテキスト・ファイル以外のテキストを探して入手し (その方法も説明せよ)、同じような解析を行なえ。

A 古典 (文書) の電子化

知的財産権についての議論が盛んである。マルチメディアやインターネットなど、コンピューターに絡んだ問題が多い。

- 著作権にも有効期限があり、それが切れた著作物は自由に利用できる。
  有効期限は、日本の場合は作家の死後 50 年。世界的にどんどん長くする傾向にある (例: イギリス 50 年 → 70 年、アメリカ 95 年. そうするのが本当に良いことなのだろうか？)。
- 既に活字になっている文書を電子化するには OCR (optical character recognition, 光学的文字認識) などが使え、作業は半自動的に行える。
- ひとたび電子化してしまえば、完全なコピーが簡単に出来る。
- 電子化された文書は、さまざまな利用が出来る。
  再び整形して本にする、読み上げソフトで音声にする、機械翻訳する、定量分析する、 etc.

諸君に考える材料を提示する意味で、 Gutenberg プロジェクトというものを紹介しよう。

A.1 Gutenberg プロジェクト

Gutenberg プロジェクトは、 1971 年にイリノイ大学の Michael Hart の思いつきからスタートしたプロジェクトで、その行動計画は次のようなものである[9]。

> もっとも利用頻度の高い 1 万タイトルの書物を、順次、電子テキスト化していって、二十一世紀のはじめの年、すなわち 2001 年の終りまでに「グーテンベルク電子公共図書館」を作り上げること

このプロジェクトの成果は CD-ROM や WWW ページ http://www.promo.net/pg/ などで公開されている。

今では、大きな機関 (アメリカ議会図書館、日本の国会図書館などの図書館) が、古典の電子化に取り組んでいるので、ボランティア・ベースの Gutenberg プロジェクトに、実際的な意味はあまりないのかもしれないが、最初に一石を投じた営みは記憶しておくべきであろう。

[9] 参考文献: 津野海太郎、本はどのように消えてゆくのか、晶文社 (1996) から引用した。

TODO　図を描こう！