テキスト処理 第2回 (2007-04-24)

田中哲
産業技術総合研究所
情報技術研究部門
akr@isc.senshu-u.ac.jp
http://staff.aist.go.jp/tanaka-akira/textprocess-2007/

今日の内容

- テキストから正規表現にマッチした行を取り出す
- そういうツールを使う
- そういうツールを作る
- それに必要なだけRubyを学ぶ

正規表現にマッチした行を取り出す

- 英単語のリスト

  A

  A's

  AOL

  AOL's

  Aachen

  Aachen's

  ...

- easy が含まれている単語

  easy

  easygoing

  greasy

  queasy

  speakeasy

  speakeasy's

  uneasy

正規表現にマッチした行を取り出す

- 英単語のリスト

A
A's
AOL
AOL's
Aachen
Aachen's
...

- qの次がuでない単語

Chongqing
Compaq
Compaq's
Esq
Esq's
Iqbal
Iqbal's
Iraq
Iraq's
Iraqi
Iraqi's
Iraqis
Qiqihar
Qiqihar's
Sq
Sq's
Urumqi
q

コンパック
HPに買収された
コンピュータ会社

イラク

イラクの
イラク人[語]

チチハル
中国の都市

英単語のリスト

- 以下の場所に置いてある
  - http://staff.aist.go.jp/tanaka-akira/textprocess-2007/words.txt
    Windows用 (改行コードが CRLF)
  - http://staff.aist.go.jp/tanaka-akira/textprocess-2007/words
    Unix 用 (改行コードが LF)
- Unix では伝統的に /usr/share/dict/words にある

マッチした行を取り出すツール

- Unix には egrep というツールがある
- Windows でも cygwin で提供されている
  http://cygwin.com/
- egrep の正規表現は Ruby に似ている
- 使いかた
  egrep 正規表現 ファイル名
- 与えられた正規表現にマッチする行を出力する

egrep を使う (1)

- easy が含まれている単語を探す

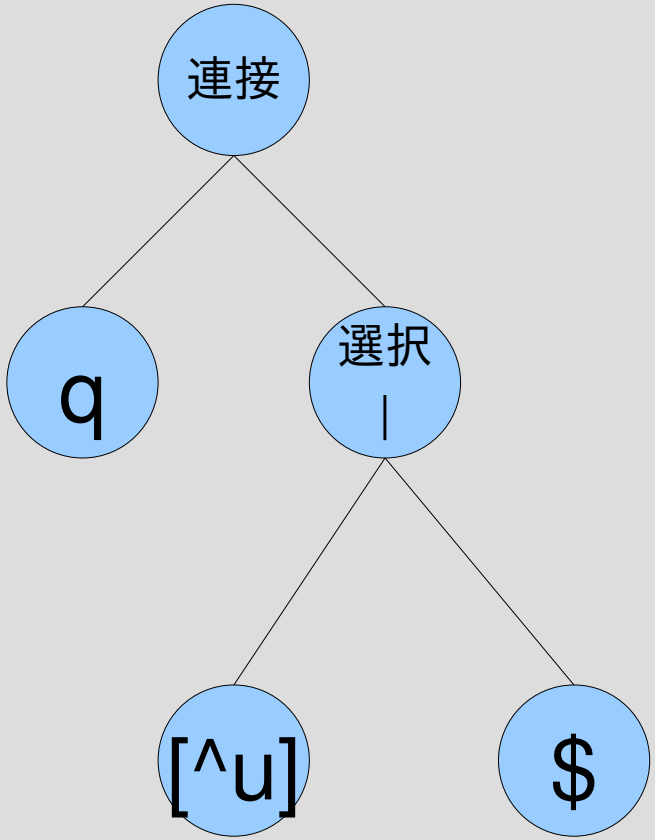
```
% egrep easy words
easy
easygoing
greasy
queasy
speakeasy
speakeasy's
uneasy
```

egrep を使う (2)

- qの次がuでない単語を探す
  - 可能性1: q の次に u でない文字が存在する
    - u でない文字を表す正規表現: [^u]
  - 可能性2: q で単語が終わっている
    - 行末を表す正規表現: $

```
% egrep 'q([^u]|$)' words
Chongqing
Compaq
Compaq's
Esq
```

q([^u]|$)



文字クラス

- 文字の集合を表す正規表現の記法
- [^u] みたいなもの: これはu以外の文字を示す
- [abc] は a, b, c のどれか
  - a|b|c と等価
- [a-z] はアルファベット小文字 26文字のどれか
  - a|b|c|...|x|y|z と等価
- [A-Za-z0-9_] はアルファベットもしくは数字もしくはアンダースコア
- [^a-z] はアルファベット小文字以外の文字
  - [^...] は [...] にマッチしない一文字にマッチする
  - 原理的にはすべての文字を列挙すれば ?|?|...|?|? と書ける

egrep を使う (3)

- qの次がuでない単語を探す
  - まずqが含まれている単語を探す
  - その結果から、quが含まれているものを除く
    - パイプで egrep をふたつつなぐ
    - -v オプションで正規表現にマッチ「しない」行を出力

```
% egrep q words | egrep -v qu
Chongqing
Compaq
Compaq's
Esq
...
```

egrep q words | egrep -v qu

- egrep 'q([^u]|$)' words と
  egrep q words | egrep -v qu は微妙に異なる
- ふたつ以上 q があり、uが後ろにあるものと後ろにないものが混ざっているとき: quqo など
- words にはそういう例はないので同じ結果になる
- qがふたつ以上入っている単語:

```
% egrep 'q.*q' words
Albuquerque
Albuquerque's
```

Albuquerque: アルバカーキ
(New Mexico州中部の都市)

大文字・小文字

- egrep 'q.*q' では Qiqihar は出てこない
- q は Q にマッチしない
  - 大文字と小文字は別の文字
- マッチさせる方法
  - q または Q にマッチする正規表現を書く
    - egrep '(q|Q).*(q|Q)' words
    - egrep '[qQ].*[qQ]' words
  - egrep の -i オプションを使う
    - egrep -i 'q.*q' words

```
% egrep -i 'q.*q'
words
Albuquerque
Albuquerque's
Qiqihar
Qiqihar's
```

egrepもどきを作る

- とりあえず -v とか -i とかのオプションは扱わない
- 処理の流れ
  - 引数に指定されたファイルをオープンする
  - 一行読む
  - ファイルの終わりで行がなければ終了
  - 引数に指定された正規表現に読み込んだ行がマッチするか調べる
  - マッチしたらその行を表示する
  - 一行読むところから繰り返す
  - ファイルをクローズする

Rubyでegrepもどきを書く

- 変数
- 代入
- 値、真偽値
- 制御構造
  - 条件分岐: if
  - 繰り返し: while
- メソッド呼び出し (関数呼び出し)
- 文字列
- 正規表現オブジェクト
- 配列
- コマンドライン引数
- ファイル

egrep.rb

```
pattern = ARGV[0]
filename = ARGV[1]

regexp =
Regexp.compile(pattern)
```

```
f = open(filename)
while line = f.gets
  if regexp =~ line
    print line
  end
end
f.close
```

egrep.rbの実行

```
% ruby egrep.rb 'q.*q' words
Albuquerque
Albuquerque's
```

変数と代入 (1)

- Rubyには変数が何種類かあるが、ここで扱うのはローカル変数
- ローカル変数に使える名前: [a-z_][A-Za-z0-9_]*
  先頭の一文字は英小文字もしくはアンダースコアで、二文字目以降は英数字もしくはアンダースコアでなければならない
  二文字目以降の長さは任意
  この規則はCとだいたい同じだが、先頭の文字が大文字であってはならないところが異なる

変数と代入 (2)

- Rubyでの書き方
  Cと違い宣言不要
  v = 10
- 変数に型がない
  v = 100
  v = "abc" 問題ない

- Cでの書き方
  int v;
  v = 10;
- 変数に型がある
  int v;
  v = 100;
  v = "abc"; はダメ

値

- Rubyの値

- 整数: 0, 1, 1000
- 浮動小数点数: 3.14
- 文字列: "abc"
- 配列: [1,2,3]
- ハッシュ: {1=>2, 3=>4}
- 正規表現: /abc/
- ユーザ定義クラス
- true, false, nil
- etc.

- Cの値

- 整数 (char, int, etc.)
- 浮動小数点数: 3.14
- ポインタ
- 配列
- 構造体
- 共用体
- etc.

真偽値

- Rubyの場合
- nil と false は偽
- それ以外は真

- 注意
  - 0は真
  - 空文字列 "" も真
  - 空配列 [] も真

- Cの場合
- 0は偽
- それ以外は真

制御構造 (1) 条件分岐

- Ruby での書き方

```
if 条件式
  真の場合のコード
else
  偽の場合のコード
end
```

- Cでの書き方

```
if (条件式) {
  真の場合のコード
} else {
  偽の場合のコード
}
```

else の省略

- Ruby での書き方

```
if 条件式
  真の場合のコード
end
```

- Cでの書き方

```
if (条件式) {
  真の場合のコード
}
```

制御構造 (2) 繰り返し

- Rubyでの書き方

```
while 条件式
  繰り返すコード
end
```

- Cでの書き方

```
while (条件式) {
  繰り返すコード
}
```

関数呼び出し

- Rubyでの書き方

  func(arg1, arg2, ...)

- 場合によってはかっこを省略可能
  func arg1, arg2, ...

- 例: ファイルオープン
  f = open(filename)

- Cでの書き方

  func(arg1, arg2, ...)

- 例: ファイルオープン
  FILE *f;
  f = fopen(filename, "r");

メソッド呼び出し

- Rubyでの書き方

  recv.meth(arg1, ...)

- 場合によってはかっこを省略可能
  recv.meth arg1, ...

- C++での書き方

  recv.meth(arg1, ...)

文字列(1) リテラル

- Rubyでの書き方

- "abc"
  Cと同様に ¥n とかを解釈する
- 'abc'
  ¥¥ と ¥' しか効かない
  'abc¥n' は 5文字

- Cでの書き方

- "abc"

文字列(2) 連結

- Rubyでの書き方

```
a = "abc"
b = "def"
c = a + b
puts c
```

- 結果は abcdef

- Cでの書き方

```
char *a = "abc";
char *b = "def";
char c[100];
strcpy(c, a);
strcat(c, b);
printf("%s¥n", c);
```

- 結果は abcdef

正規表現オブジェクト

- リテラルの書き方
  /正規表現/
- 文字列にマッチさせる
  正規表現オブジェクト =~ 文字列
- 文字列から正規表現オブジェクトを作る
  Regexp.compile(文字列)
  (Regexp というオブジェクトに対する compile というメソッド呼び出し)

配列

- Rubyでの書き方

  ary = [要素1, 要素2, ...]

- 要素の取り出し
  ary[添字]

- 要素の更新
  ary[添字] = 式
  存在しない場所を更新すれば自動的に配列の長さが伸びる

- Cでの書き方

  int ary[] = { 要素1, ... };

- 要素の取り出し
  ary[添字]

- 要素の更新
  ary[添字] = 式

コマンドライン引数

- Rubyでの書き方

  ARGV[0] のように参照

- ARGVという配列が用意されている

- ARGV[0]が最初の引数

- Cでの書き方

```
int main(
    int argc,
    char **argv) {
  argv[1] のように参照
}
```

- argv[1] が最初の引数

ファイル

- Rubyでの書き方
```
f = open(name)
while line = f.gets
  ...
end
f.close
```
- open() でオープン
- f.gets で一行読み込み
- f.close でクローズ

- Cでの書き方
```
FILE *f = fopen(name, “r”);
char line[4096];
while (fgets(line, 4096, f)
                != NULL) {
  ...
}
fclose(f);
```

コメント

- Rubyでの書き方

  # コメントの内容

- # から行末までがコメントになる

- Cでの書き方

  /* コメントの内容 */

- /* と */ の間がコメントになる

egrep.rb

```
pattern = ARGV[0]              # 第1引数の取り出し
filename = ARGV[1]             # 第2引数の取り出し
regexp = Regexp.compile(pattern)    # 第2引数の文字列を
                                    # 正規表現オブジェクトに変換
f = open(filename)   # 第1引数のファイルをオープン
while line = f.gets    # ファイルの終わりまで一行づつ読む
  if regexp =~ line    # 読み込んだ行は正規表現にマッチするか？
    print line         # マッチしてたら表示
  end
end
f.close                # ファイルをクローズ
```

Rubyらしい?

- じつはegrep.rb はあまり Rubyらしくない
- Rubyらしく書くともっと短くなる

```
pattern = ARGV.shift
regexp = Regexp.compile(pattern)
ARGF.each {|line|
  print line if regexp =~ line
}
```

- こういう書き方については次回

まとめ

- words を例に egrep を使ってみた
- egrep もどきを Ruby で作ってみた