テキスト処理'15 ＃5 – (実用的なデータ処理)

久野　靖 *

2015.6.9

1　本日の内容

いよいよ最終回ですが、本日の内容はやり残していることが 3 つあるので、それらを扱います。少し忙しいかも知れませんが、頑張りましょう。

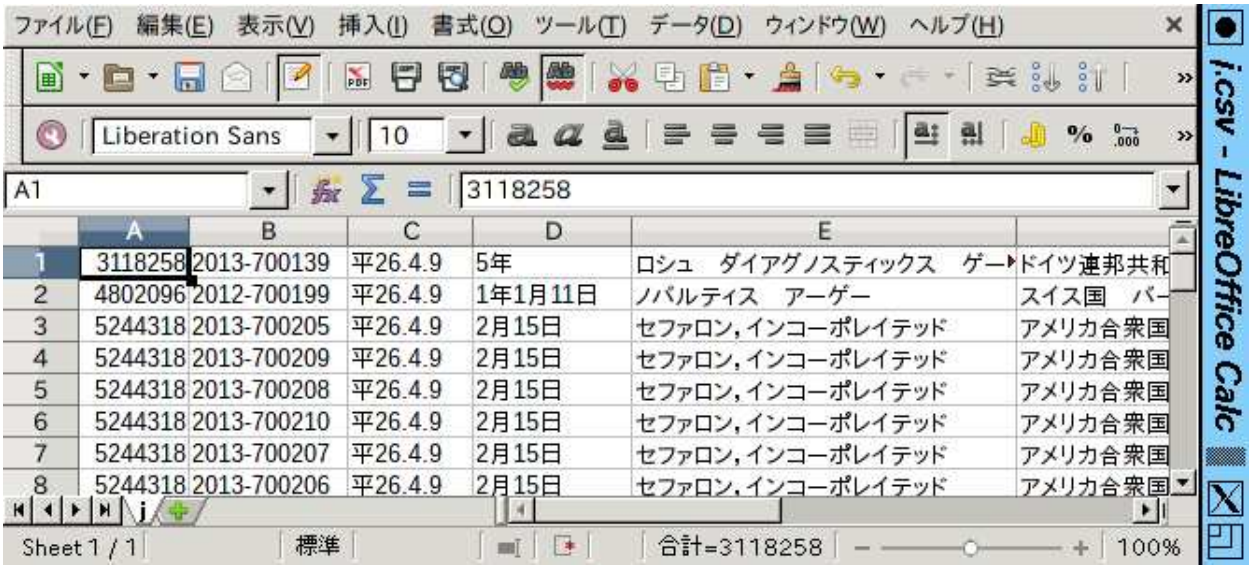
(1) スクレイピングデータの CSV/TSV への書き出し

(2) 複数テーブルのハッシュによる突き合わせ

(3) 名寄せ問題

2　スクレイピングデータの CSV への書き出し

前回のスクレイピングでは、画面表示までででファイルに書いていませんでした。今回は特許データを例にとって、内容をファイルに書くところまで行います。図 1 に、出力した CSV を表計算ソフトに読み込んだようすを示します。



	A	B	C	D	E	
1	3118258	2013-700139	平26.4.9	5年	ロシュ　ダイアグノスティックス　ゲー	ドイツ連邦共和
2	4802096	2012-700199	平26.4.9	1年1月11日	ノバルティス　アーゲー	スイス国　パー
3	5244318	2013-700205	平26.4.9	2月15日	セファロン, インコーポレイテッド	アメリカ合衆国
4	5244318	2013-700209	平26.4.9	2月15日	セファロン, インコーポレイテッド	アメリカ合衆国
5	5244318	2013-700208	平26.4.9	2月15日	セファロン, インコーポレイテッド	アメリカ合衆国
6	5244318	2013-700210	平26.4.9	2月15日	セファロン, インコーポレイテッド	アメリカ合衆国
7	5244318	2013-700207	平26.4.9	2月15日	セファロン, インコーポレイテッド	アメリカ合衆国
8	5244318	2013-700206	平26.4.9	2月15日	セファロン, インコーポレイテッド	アメリカ合衆国

図 1: 抽出データを表計算ソフトで読んだところ

では実際にプログラムを見てみましょう。**注意!:** 最初に注意すべきことですが、前回説明したように `Nokogiri::HTML.parse` を呼ぶときに正しい文字コードを指定しないと解析できません。今回のファイルは **HTML** 冒頭を見ると **shift-jis** と書かれているので、そのように指定しています。

*経営システム科学専攻

```
require 'open-uri'
require 'nokogiri'
require 'csv'

$url = 'http://w3in/~kuno/tproc15/QUERY_files/wBl7lUalist_date.html'
$proxy = { :proxy => 'http://w3proxy:8080/' }
$html = open($url, $proxy) do |f| f.read end
$page = Nokogiri::HTML.parse($html, nil, 'shift-jis')
$code = 'euc-jp'

def extract
  CSV.open('j.csv', 'wb') do |f|
# open('j.tsv', 'wb') do |f|
    tbls = $page.search('//table[position()>1]')
    tbls.each do |tbl|
      r = []
      num =  tbl.search('./tbody/tr[1]/td[2]/a/text()').to_s.slice(/\d+/)
      r.push(num)
      r.push(tbl.search('./tbody/tr[1]/td[3]/text()').to_s.encode($code))
      r.push(tbl.search('./tbody/tr[1]/td[4]/text()').to_s.encode($code))
      r.push(tbl.search('./tbody/tr[1]/td[5]/text()').to_s.encode($code))
      r.push(tbl.search('./tbody/tr[3]/td[2]/text()').to_s.encode($code))
      r.push(tbl.search('./tbody/tr[3]/td[3]/text()').to_s.encode($code))
      f << r
#     f.puts(r.join("\t"))
      txt = tbl.search('./tbody/tr[5]/td[2]/pre').to_s.encode($code)
      txt.gsub!(/<br>/, "\n")
      txt.gsub!(/<pre>|<\/pre>/, "")
      open("j#{num}.txt", "wb") do |f1| f1.puts(txt) end
    end
  end
end
extract
```

では既に知っていることも多いと思いますが、ひととおり説明します。

- 冒頭でライブラリとして open-uri、nokogiri、そして CSV 書き出しのための csv を指定しています。
- URL は前回説明したように frameset タグの中をみてデータ本体がある場所を調べ、そこを指定しています。[1]
- `$html` には URL から HTML をまとめて取ります。外部に直接出られるところで実行する場合は`$proxy` は指定不要です。
- `$page` は Nokogiri で解析した結果でした。文字コード指定については前述の通り。
- 出力文字コードを何箇所かで指定するので、それを`$code` に入れておきます (1 箇所で変更できるように)。
- `extract` が作業用をおこなうメソッドです。
- まず CSV 出力用のファイルを開きますが、この場合は CSV ライブラリの open を呼びます。TSV の場合は次のコメントアウトしてある行を使いますが、こちらの場合は普通の open です。
- 次に、特許の各データを順次処理するため、位置が 1 より大きい (先頭は見出しのみ)TABLE 要素の集まりを取得します。
- この先で each ループを使い、その個々の TABLE 要素を変数 `tbl` に取り出しながら作業します。
- まず、その特許データを 1 つの行として CSV/TSV に出力するので、それを用意するための配列 `r` を用意しておきます。

[1] GSSM 外部から試す場合は「`http://www2.gssm.otsuka.tsukuba.ac.jp/staff/kuno/tproc15/...`」の URL を使ってください。

- この先ですが、各 TABLE 要素「より中の」ものを取るので、こんどは tbl に対して.search を実行し、そこで XPath 式としては「ここ (.) から下」の指定を使います。
- まず特許番号を変数 num に取ります。これは先頭の TR の下の 2 番目の TD で、そのテキスト部分だけ取り出し、to_s で文字列にします。最初はこれでいいと思ったのですが、なぜか空白文字がくっついていたので、文字列に対するメソッド slice で「残す文字のパターン」を指定し、数字列のみ残るようにしました。
- その先、出願番号、年月日、期間、申請者名、申請者所在はそれぞれぞれ同様の XPath 式で取っています。これらは日本語文字を含むので.encode で文字コードを揃えます。
- 以上で 1 行ぶんのデータができたので、CSV の場合は「f << r」で書き出します。TSV の場合は配列の各要素をタブをはさんでくっつけるのに文字列のメソッド.join を使い、出力は f.puts でおこないます。
- 処分の内容は複数行のテキストなので、「j 特許番号.txt」というファイルに特許ごとに書くことにします。最初は XPath 式で text() を指定してテキスト化していたのですが、それだと改行 (
タグ) が消えてしまうので、text() は指定せずに文字列化し、その後で
は改行文字に変換、また<pre>と</pre>は空文字列に変換しています。最後に上述のファイルを開き、そこに書き出しています。

演習 30 演習 28 や演習 29 でやった、自分なりに取り出したものについて、CSV または TSV で書き出し、表計算ソフトに読めることを確認しなさい。

3 複数テーブルの突き合わせ

3.1 レコード型

次は再び＃ 3 で扱ったパネルデータの話題に戻りますが、扱い方をいくらか変えます。まず、＃ 3 ではパネルデータを配列に読んで扱っていましたが、それだと「どのカラム」を全部番号で言うことになるので、間違いが起きやすくなります。そこで、各データ項目を番号でなく名前で指定する方法である「レコード」について説明しましょう。なお、以下の書き方は Ruby ですが、レコードそのものは他の言語にもあります。

まず、今回は項目マスターと個別売り上げを 2 つの表に分けてあり、CSV ファイルも 2 つになっています。その名前と内容は次の通りです。

- IMASTER2.csv — アイテム ID、商品名、分類名
- ISALES2.csv — パネル ID、店舗 ID、レジ ID、レシート ID、時刻、アイテム ID、品数、合計金額

これらを名前で扱うために、Item および Sale という名前のレコード型を次のように定義します。

```
Item = Struct.new(:item, :name, :cat)
Sale = Struct.new(:pane, :shop, :reg, :rec, :time, :item, :num, :amt)
```

実際に個々の項目データや売り上げデータを作るときは、次のようにします。値の並び順は Struct.new で指定した順です。

```
i = Item.new(値, 値, 値)
s = Sale.new(値, 値, 値, 値, 値, 値, 値, 値)
```

あとは、たとえば「i.name」で商品名、「s.time」で時刻が参照できます (配列の要素と同様、書き換えることもできます)。

3.2 例題: 分類ごとの売り上げ集計

では今回の例題ですが、商品ごとにその名前だけでなく分類が記されていますから (このように表を分けたことで、商品ごとのより詳しい情報を入れる場所ができたことに注意)、どの分類がいくら売れたかを集計してみましょう。プログラムを示します。

```
require 'csv'
Item = Struct.new(:item, :name, :cat)
Sale = Struct.new(:pane, :shop, :reg, :rec, :time, :item, :num, :amt)

def readdata
  $items = {}
  CSV.foreach('IMASTER2.csv') do |r|
    $items[r[0].to_i] = Item.new(r[0].to_i, r[1], r[2])
  end
  $sales = []
  CSV.foreach('ISALES2.csv') do |r|
    $sales.push(Sale.new(r[0].to_i, r[1].to_i, r[2].to_i, r[3].to_i,
                         r[4].to_i, r[5].to_i, r[6].to_i, r[7].to_i))
  end
end
def cat_summary
  sum = Hash.new(0)
  $sales.each do |s|
    cat = $items[s.item].cat
    sum[cat] += s.amt
  end
  r = []
  sum.each do |cat, amt| r.push([cat, amt]) end
  r.sort! do |x, y| x[0] <=> y[0] end
  r.each do |x| printf("%8d %s\n", x[1], x[0]) end
end
readdata
cat_summary
```

- readdata は例によってファイルからデータを読みます。今回は読んだデータを直接グローバル変数に格納しているので、とくに値は返しません。
- 今回は CSV を読み込むのには CSV ライブラリの機能を使いました。CSV.foreach を使うと、CSV の各行を 1 つの配列として繰り返しブロックに渡してくれます。
- 商品の情報は商品 ID をキーとしてハッシュ$items に格納します。個々のデータは Item レコードです。
- 個別の売り上げは Sale レコードで、これはファイルから読んだ順番に配列に並べています。
- cat_summary が分類ごとの金額を集計して表示するメソッドです。
- sum にハッシュを用意します。
- 各売り上げごとに、その商品 ID から分類名を取り出し、続いてその分類名の欄の合計を金額ぶん増やします。
- 累計が終わったら結果データを入れる配列 r を用意し、ハッシュの内容 (分類名、金額) を 2 要素の配列としたものを r にすべて追加します。
- 分類名順が見やすいでしょうから、分類名順に整列します。
- 最後に、r の内容を表示します。

演習 31 このプログラムを動かしてみなさい。動いたら、次のことをするプログラムを紙に書きなさい。

a. 分類ごとの売り上げ点数 (商品数の合計) を表示。★
b. 分類ごとの売り上げ件数 (個数に関わらず売れた回数) を表示。★
c. 分類ごとにその分類中の最も売れた合計額の高い商品名を表示。
d. 店舗ごとの各分類の売り上げ金額合計を表示。
e. 時刻ごとの各分類の売り上げ件数を表示。

演習 32 前問の演習の結果 (どれでも) を表示するかわりに CSV に出力して表計算ソフトに読んでみなさい。次に、このパネルデータから「興味深い」分析を何か行ってみなさい。分析方法としては Ruby で集計分析してもいいですし、表計算ソフト上で分析する方が好きならそれでもよいです (両方組み合わせるとなおよいでしょう)。

4 名寄せの問題

ずっと後回しになっていましたが、名寄せの問題です。とある会社の特許のデータを扱います。

```
#出願人, 番号, 発明の名称, 出願日, 発明者 1, 発明者 2, 発明者 3, 発明者 4, 発
明者 5, 発明者 6, 発明者 7, 発明者 8, 発明者 9, 発明者 10, 発明者 11, 発明者
12, 発明者 13, 発明者 14, 発明者 15, 発明者 16, 発明者 17, 発明者 18, 発明者
19, 発明者 20, 発明者 21, 発明者 22, 発明者 23, 発明者 24, 発明者 25, 発明者
26,UPC 1,UPC 2,UPC 3,UPC 4,UPC 5,UPC 6,UPC 7
Apple Inc.,USD714294,Housing plate,2012.08.17,Dinh; Richard Hung
Minh,Howarth; Richard P.,Myers; Scott A.,Pakula; David A.,Tan;
Tang Yew,,,,,,,,,,,,,,,,,,,,,,,,,D14/439,,,,,,
Apple Inc.,USD713855,Display screen or portion thereof with
graphical user interface,2012.03.06,Roberts; Samuel
Morgan,Ubillos; Randall,,,,,,,,,,,,,,,,,,,,,,,,,D14/487,,,,,,
...
```

ここで知りたいのは、この会社で誰が何件の特許の (共同) 出願者となっているか、ということだとします。それには、1 行ずつ CSV を読んで、特許番号と出願者の組を抽出していきます。まずは名前を整列して順に表示するプログラムを見てみましょう。

```
require "csv"
def readdata
  r = []
  num = 0
  CSV.foreach("PAT1.csv") do |a|
    num += 1; if num == 1 then next end
    for i in 4..29 do
      if a[i] !=~ /^$/ then r.push([a[1], a[i]]) end
    end
  end
  return r
end
def countbynames(a)
  h = Hash.new(0)
  a.each do |x| h[x[1]] += 1 end
  return h
end
def printnames(h)
  h.keys.sort.each do |name| puts(name) end
end
$id_name = readdata
$name_count = countbynames($id_name)
printnames($name_count)
```

次に、各特許ごとに発明者 1〜26 は配列の 4〜29 番になるので、それを取り出し、空でないなら「特許番号、名前」の組を結果に追加していきます。ここまでが readdata の処理です。

次の countbynames はハッシュ表を使い、各人が何件の特許に加わっているかを累計していきます。printnames はハッシュ表からキーの並びの配列を取り出し、整列して表示しています。これをやった冒頭部分を見てみましょう。

```
Adam; Chris
Adams; Steven
Ai; Jiang
Akahane; Ryosuke
Akana; Jody
Akana; Jody Be
Alben; Lauralee A.
Amelse; Adrian J.
Andre ; Bartley K.
Andre; Bart K.
Andre; Bartley K
Andre; Bartley K.
Andre;Bartley K.
Andreini; David
```

これを見ると、同一人物が複数回現れていることが分かります。それはなぜかというと、同一人物でも名前の表記に微妙な揺れがあり、そのため文字列としては同一になっていないからです。

人間が眺めるぶんにはこれでも問題はないのですが、プログラムの処理により一人一人の性質を集約しようとすると (今回の場合は誰が何件の特許に関与したかですね)、このままではまずいわけです。そこで、表記の揺らぎを吸収して「一人のデータは一人として扱う」ようにしたいわけです。このような、さまざまな経緯で分かれてしまっている同一人 (や組織) のデータを統合することを「名寄せ (agglegation)」と言います。

名寄せの 1 つの方法は、人間が実際に目でみて違うものを統一した形に修正することです。たとえば、次のようなファイルを手で作るとします。

```
$normalize = {
'Akana; Jody' => 'Akana; Jody Be',
'Andre ; Bartley K.' => 'Andre; Bartley K.',
'Andre; Bart K.' => 'Andre; Bartley K.',
'Andre; Bartley K' => 'Andre; Bartley K.',
'Andre;Bartley K.' => 'Andre; Bartley K.',
...
}
```

このようなファイルがあったとすれば、それをプログラムの一部に組み込んでおき、名前を読み込んだ直後に次のようにして表記のゆれを統一できます。

```
if $normalize[name] != nil then name = $normalize[name] end
```

ここで問題は、目で見て同じ名前の書き換え表を作るみたいな単純労働は人間がやるべきことではない、ということです。そこで今回は、ある程度までコンピュータを使って名寄せをサポートすることを考えてみます。

5　類似度の判別

要するに今回必要なのは、「全く同一ではないが、類似している文字列について、類似していると判断する」ことです。そのような 1 つの方法として、アラインメント (alignment、対応づけの計算) があります。たとえば、「akasaka」「sakasama」「isasaka」という 3 つの語で、互いにどれとどれが「一番似て」いるでしょうか?

図 2: 対応つけの例

1 つの考え方として、図 2 のように「対応のつくもの」どうしを線で結ぶとして、「交差しないように引ける線の最大本数」で類似度を計ることができます。問題は、さまざまな線の引き方のうち最大をどうやって求めるかですが、これは「動的計画法」というやり方を使えばできます。以下に説明しましょう。

		s	a	k	a	s	a	m	a
	0	1	2	3	4	5	6	7	8
a	1								
k	2								
a	3								
s	4								
a	5								
k	6								
a	7								

		s	a	k	a	s	a	m	a
	0	1	2	3	4	5	6	7	8
a	1	2	1	2	3	4	5	6	7
k	2	3	2	1	2	3	4	5	6
a	3	4	3	2	1	2	3	4	5
s	4	3	4	3	2	1	2	3	4
a	5	4	3	4	3	2	1	2	3
k	6	5	4	3	4	3	2	2	3
a	7	6	5	4	3	4	3	3	2

図 3: アラインメントの動的計画法

2 次元配列を用い、縦と横にそれぞれ文字列の文字を 1 文字ずつ配置します (一番先頭はあけてあります)。縦と横の位置はそれぞれ、その文字列をどこまで進んだかを意味します。両方の文字列全体を対応させるのですから、最後は右下隅のところまで来ます。

これから、この配列の中に「2 つの (部分) 文字列を対応させるのに、文字列の挿入・削除を何回行ったか」を数えて行きます。まず、左上隅はどちらの文字列も 1 文字も進んでないので、挿入・削除は 0 回なので 0 と記入します。そして上端の列、左端の列は 1 文字進むごとに 1 文字ずつ挿入・削除するので 1 ずつ増える値を記入します (図 3 左)。一般に、右に 1 マス、または下に 1 マス進むということは、その文字を飛ばすことですから 1 個の挿入・削除になります。

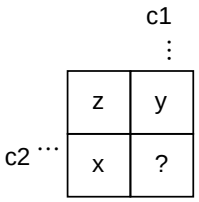


図 4: アラインメントの値の計算方法

さて、ここからが大切なところです。図 4 のような状況で、「?」のますの値を記入したいとします。ここで x、y、z はもう値が求まっているものとします。そうすると、次の 3 つ場合があり得ます。

- x のマスから右に 1 つ動く。値は x+1。
- y のマスから下に 1 つ動く。値は y+1。
- z のマスから右下に 1 つ動く。値は、「そのマスに対応する縦横 2 つの文字 c1 と c2 が同じ文字」ならそのまま対応するので z でよい。「同じでない」なら、c1 を削除して c2 を挿入するので z+2。

ここで目的は「できるだけ対応する文字を多くしたい (つまり挿入削除を少なくしたい)」わけですから、上の 3 つの場合のうち「最も小さい値」を「?」のところに記入すればよいわけです。これを左上から右下まですらっと順番に行うと、右下隅のマスに値が求まり、これが 2 つの文字列全体を対応させるときの「できるだけ少ない挿入・削除の回数」になります (なお、この数値のことを「編集距離」と呼びます)。

では、2 つの文字列を渡されて、編集距離を返すメソッドを示します (badness については後で)。

```
def editdist(s, t)
  a = Array.new(s.size+1) do |i| Array.new(t.size+1) do |j| i+j end end
  for i in 1..s.size do
    for j in 1..t.size do
      x = a[i-1][j-1]
      if s[i-1] != t[j-1] then x += 2 end
      a[i][j] = [a[i-1][j]+1, a[i][j-1]+1, x].min
    end
  end
  return a[s.size][t.size]
end
def badness(s, t)
  return editdist(s, t) - (s.size - t.size).abs
end
```

これを動かしてみましょう。

```
irb> load 'editdist.rb'
  => true
irb> editdist 'akasaka', 'sakasama'
  => 3
irb> editdist 'sakasama', 'isasaka'
  => 5
irb> editdist 'isasaka', 'akasaka'
  => 4
```

挿入・削除は 1 点、違う文字どうしの対応は挿入+削除なので 2 点であることに注意。

ところで、後ろにくっついている badness は何でしょう? 今やりたいのは、英語の人名の「近さ」を調べたいのだということに注意。そうすると、人名の表記の揺れは「Steven」と「Steve」や「S」のように一部を省略する形が多いことが分かります。省略をすると文字列は短くなりますから、このような関係にある 2 つの文字列では「編集距離 (挿入・削除の数) から文字列の長さの差を引く」と 0 になるはずです。これを計算するのが badness なわけです。

6 名寄せのためのハッシュ表生成

では、実際に前述のようなハッシュのリテラルを書き出すプログラムを見てみましょう。先に生成した名前の並びが `names.list` というファイルに入っているものとして、それを読んで処理します。

このとき、直接名前の文字列を badness で判定するのではなく、その文字列をコピーしてから「大文字を小文字に直す」「ピリオドを削除」などいくつかの作業をおこなって「正規化」してから処理に入るようにしています。その方が何かと柔軟性が高いので。

```
def readnames
  r = []
  open('names.list') do |f|
    f.each_line do |s| r.push(s.chop) end
  end
  $names = r
end
def makecanon
  b = Array.new($names)
  b.each_index do |i|
    b[i] = $names[i].dup
    b[i].tr!('A-Z', 'a-z')
    b[i].sub!(/ +/, ' ')
    b[i].sub!('.', '')
    b[i].sub!(/^ +/, '')
    b[i].sub!(/ +$/, '')
  end
  $canon = b
end
 //editdist, badness here...
def checkcomb
  a = $canon
  $map = Array.new(a.size, -1)
  b = Array.new(a.size, 99)
  for i in 0..a.size-2 do
    for j in i+1..a.size-1 do
      x = badness(a[i], a[j])
      if x < 4 && b[j] > x then $map[j] = i; b[j] = x end
    end
  end
end
def printmap
  puts "$normalize = {"
  $names.each_index do |i|
    k = $map[i]; if k < 0 then next end
    while $map[k] >= 0 do k = $map[k] end
    puts("  '#{$names[i]}' => '#{$names[k]}', // #{k}")
  end
  puts("}")
end
 readnames
 makecanon
 checkcomb
 printmap
```

readnames はファイルから名前一覧を配列$names に読み込みます。makecanon は名前一覧のコピーを作り、大文字を小文字に変換、複数の空白を 1 つに、ピリオド削除、行頭行末の空白削除などを行い、配列$canon に入れます。checkcomb はすべての名前の組について badness を呼んで値が 4 未満なら$map に書き換えるべき相手の番号を入れます (初期値は -1)。最後に printmap は各名前について$map の値が -1 でないならハッシュの項目を生成します (なぜ while ループがあるかというと、A は B、B は C にのような書き換えは全部たどってから生成するため)。これを使って生成した結果は次のような感じです。

```
$normalize = {
  'Akana; Jody Be' => 'Akana; Jody', // 4
  'Andre; Bart K.' => 'Andre ; Bartley K.', // 8
```

```
    'Andre; Bartley K' => 'Andre ; Bartley K.', // 8
    'Andre; Bartley K.' => 'Andre ; Bartley K.', // 8
    'Andre;Bartley K.' => 'Andre ; Bartley K.', // 8
    'Batailou; Jeremy' => 'Bataillou; Jeremy', // 24
    ...
}
```

演習 22 上記のプログラムをコピーして動かしてみなさい。動いたら今度は「名寄せに加わらなかった名前の一覧」を表示してみなさい (または Unix コマンドでやってもよい)。その上で、さらに名寄せをうまくやるにはどういう工夫を加えたらいいと思うか、検討しなさい。

演習 23 演習 8 の検討結果を実装し、評価しなさい。

演習 24 名寄せプログラムは耐えられないとまでは言わないが、かなり遅い。もし badness の限界を 4 とするのなら、予め「2 つの文字列の長さの差+4」を editdist に与えることにより、これよりも値が大きくなりそうならすぐにやめて戻るようにすることで、時間を短縮できるはずである。具体的にどのようにしたらいいと思うか、検討しなさい。

演習 25 演習 10 の検討結果を実装し、評価しなさい。

7 第 5 回レポート課題

2 つ以上の演習のプログラムを選んで「プログラムを実際に動かす」課題をおこない、レポートとして提出しなさい。レポートではプログラムごとに次の内容を含むこと。

- 学籍番号、氏名、提出日付 (これらは最初に 1 回でよい)
- 問題の再掲
- 作成したプログラムとその説明
- 実行結果と考察

とくに考察において、「ロジックのどこが難しいか、どのように工夫したか」をきちんと書くこと。レポート課題は次回授業開始時までに、`gssm.k-kiso` に投稿すること。ただし★がついている問題 (易しい問題) を含む場合はは本日中。なお、最終レポートなので、「できれば」それなりに興味深い問題で、興味深い考察があることを希望します。