Conversion del report de medicacion

V0

Limpiando el archivo HTML

La herramienta de report devuelve un archivo html bastante sucio,

```
.
.
.

actament_actiu_SN


</th><th style="color:black;border:0;background:transparent;text-
align:left;vertical-align:middle;">{...}</th></tr><t
r><th style="color:black;border:0;background:transparent;text-
align:left;vertical-align:middle;">1</th><td
style="color:black;border:0;background:#dcdcc8;text-align:left;vertical-
align:middle;">zyuprexa 5 mg/nit, rivotril 0.5 mg /nit, exelon
parche,..</td><td style="color:black;border:0;background:transparent;text-
align:right;vertical-align:middle;">zyuprexa=1.000, rivotril=1.000,
exelon=1.000, Interno=20100303.0</td></tr><tr><th
style="color:black;border:0;background:transparent;text-align:left;vertical-
align:middle;">2</th><td
style="color:black;border:0;background:#dcdcc8;text-align:left;vertical-
align:middle;">zyprexa 5mg-0-10mg


rivotril 0,5mg -1-1-1


depakine 200 1-0-1


orfidal 0-0-1 ??
.
.
.
```

que aun se podria *parsear* con un poco de esmero pero lo mas sencillo es limpiar todo esto. Voy a hacerlo con **html2text** pero dejando el codigo html, sobre todo para obviarlo ;-P

```
$ html2text -unparse -o toparse.html oscar1.html
```

y la cosa mejora bastante,

```
.
.
.
<TH style="color:black;border:0;background:transparent;text-
align:left;vertical-align:middle;">
<P>
1
</P>
</TH>
<TD style="color:black;border:0;background:#dcdcc8;text-align:left;vertical-
align:middle;">
<P>
zyuprexa 5 mg/nit, rivotril 0.5 mg /nit, exelon parche,..
</P>
</TD>
<TD style="color:black;border:0;background:transparent;text-
align:right;vertical-align:middle;">
<P>
zyuprexa=1.000, rivotril=1.000, exelon=1.000, Interno=20100303.0
</P>
</TD>
</TR>
.
.
.
```

De hecho mejora tanto que ahora todas las lineas que me interesan estan ya separadas. Lo unico que he de hacer es quedarme con las lineas que contengan **Interno=**.

```
$ grep "Interno=" toparse.html > toparse.txt
```

```
.
.
.
zolpidem=1.000, semandrol=1.000, pravastatina=1.000, esertia=1.000,
Interno=20091211.0
zolpidem=1.000, uniket=1.000, tpm=1.000, tanakene=1.000, spiriva=1.000,
rivotril=1.000, pramipexol=1.000, plantaben=1.000, nopiafren=1.000,
nochye=1.000, nioche=1.000, ncohe=1.000, manana=1.000, keppra=1.000,
galantamina=1.000, deprax=1.000, d=1.000, comp=1.000, bernerva=1.000,
atorvastatina=1.000, arafaxina=1.000, adiro=1.000, Interno=20111422.0
zolpidem=1.000, sp=1.000, lormetazepam=1.000, hidroxil=1.000, folico=1.000,
escitalopram=1.000, enalapril=1.000, adiro=1.000, ac=1.000,
Interno=20100777.0
zolpidem=1.000, escitalopram=1.000, ebixa=1.000, ditropamn=1.000,
Interno=20100325.0
zolpidem=1.000, cymbalta=1.000, alprazoalm=1.000, Interno=20100563.0
zolpidem=1.000, paroxetina=1.000, lyrica=1.000, ibuprofeno=1.000,
hidroferol=1.000, fosamax=1.000, duloxetina=1.000, alprazolam=1.000,
Interno=20170607.0
.
```

```
.
.
```

Ya de aqui se puede empezar a trabajar

Parsing

La idea general es leer el id del sujeto *Interno=XXXXXX.0* y crear una estructura por cada sujeto donde se almacenen los distintos medicamentos que toman. El total de todos los medicamentos posibles debe ser almacenado en otra estructura. 

Claramente, los sujetos se almacenan mejor en un *hash de hashes* asignando un valor positivo al leerlo. La lista de medicamentos, podria almacenarla en un *array*, pero entonces tendria que chequear todo al *array* antes de añadir una nueva. Lo mas sencillo es hacerlo en otro *hash* y al escribir cruzo los *hashes* y solo tengo que chequear si el medicamento existe para el sujeto correspondiente. 

Venga que no es tan dificil,

parser.pl

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

use strict;
use warnings;

my $ifile = "toparse.txt";
my $ofile = "med0.csv";

my %meds;
my %patients;

open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
    (my $gay_id) = /.*Interno=(\d*)\.0/;
    my @list = split /, /;
    foreach my $nomed (@list){
        (my $med, my $blah) = split /=/,$nomed;
        unless ($med eq "Interno"){
                $meds{$med} = 1;
                $patients{$gay_id}{$med} = 1;
        }
    }
}
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno";
foreach my $med (sort keys %meds){
```

```
    print ODF ",$med";
}
print ODF "\n";
foreach my $patient (sort keys %patients){
    print ODF "$patient";
    foreach my $med (sort keys %meds){
        if(exists($patients{$patient}{$med})){
            print ODF ",1";
        }else{
            print ODF ",0";
        }
    }
    print ODF "\n";
}
close ODF;
```

Esto es una prueba de concepto asi que no tiene escalabilidad ninguna, no hay argumentos de comando, sobreescribe siempre el mismo file, etc. Pero funciona que te cagas.

Proxima version: añadir ficheros de entrada y salida como argumentos de comando

V1

Debemos convertir un archivo excel que contiene el codigo interno y una lista de la medicacion.Una vez convertido a CSV luce así,

```
fecha;interno;medica_obtenidatextmining
40806.0;20100303.0;zyuprexa, rivotril, exelon
40815.0;20080254.0;zyprexa, sertralina, rivotril, orfidal, depakine
41730.0;20130471.0;zyprexa, ventolin, venlafaxina, trankimazin, singulair,
seretide, quetiapina, omeprazol, evopad, dia, alprazolam
41654.0;20080254.0;zyprexa, sertralina, rivotril, prometax, depakine
41058.0;20080254.0;zyprexa, sertralina, rivotril, niveles, en, depakine,
bajos
40581.0;20091211.0;zolpidem, semandrol, pravastatina, esertia
41575.0;20111422.0;zolpidem, uniket, tpm, tanakene, spiriva, rivotril,
pramipexol, plantaben, nopiafren, nochye, nioche, ncohe, manana, keppra,
galantamina, deprax, d, comp, bernerva, atorvastatina, arafaxina, adiro
40966.0;20100777.0;zolpidem, sp, lormetazepam, hidroxil, folico,
escitalopram, enalapril, adiro, ac
41754.0;20100325.0;zolpidem, escitalopram, ebixa, ditropamn
....
```

Basicamente capturamos el codigo y el texto plano en dos variables, usando los “;” como separadores. Luego el texto plano se divide en un *array* usando “,” como separador. Los nombres se almacenan como *keys* de un *hash* y se añaden tambien al sujeto como *keys* de un *hash* de *hashes*. Luego se exportan a un CSV matricialmente, cruzando los *hashes*.

un poco distinto pero basicamente lo mismo que antes,

parser.pl

```
#!/usr/bin/perl

use strict;
use warnings;

my $ifile = "medica.csv";
my $ofile = "med0.csv";

my %meds;
my %patients;

open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
    (my $gay_id, my $ldata) = /^.*;(\d*)\.0;(.*)$/;
    if ($ldata){
        my @list = split /, /, $ldata;
        foreach my $med (@list){
            if ($med) {
                $meds{$med} = 1;
                $patients{$gay_id}{$med} = 1;
            }
        }
    }
}
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno";
foreach my $med (sort keys %meds){
    print ODF ",$med";
}
print ODF "\n";
foreach my $patient (sort keys %patients){
    print ODF "$patient";
    foreach my $med (sort keys %meds){
        if(exists($patients{$patient}{$med})){
            print ODF ",1";
        }else{
            print ODF ",0";
        }
    }
    print ODF "\n";
}
close ODF;
```

Basicamente la matriz es muy grande para procesar. Hay muchos *typos* y añaden un monton de filas y columnas de porqueria.

V2

La matriz actual no es usable, pero si, *a priori*, pudieran unificarse las palabras mal escritas bajo la correcta, es de esperar que la cantidad de columnas disminuya bastante. Idealmente escribiriamos las reglas correctas a mano, revisadas y sin equivocarnos . En realidad son demasiadas palabras y la cantidad de errores es brutal. Pero vamos a ver si podemos hacer algo automagicamente.

La idea general es que casi siempre se escribe bien (de nuevo,) y que las palabras erroneas no se repiten mucho. Asi que las palabras que tienen menos repeticiones seran errores de escritura y deberiamos poder agruparlas bajo alguna regla.

Haciendo reglas

¿Cuanto se repiten las palabras?

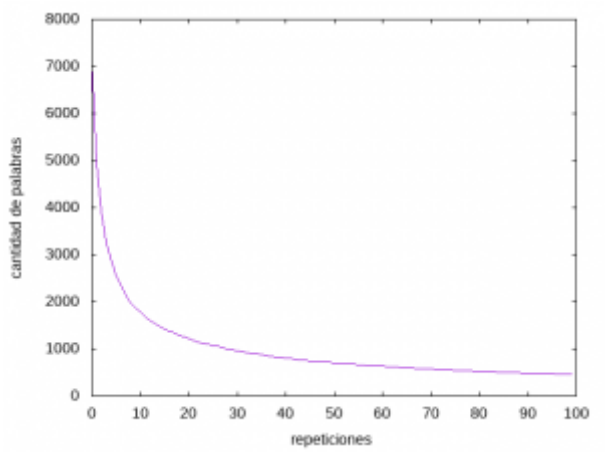
Vamos a calcular cuanto se repite cada palabra a ver si podemos hacer una reglas.

un programita rapido

rank.pl

```
#!/usr/bin/perl

use strict;
use warnings;

my $ifile = "medica.csv";
my $ofile = "medrank.csv";

my %meds;
my %patients;

open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
    (my $gay_id, my $ldata) = /^.*;(\d*)\.0;(.*)$/;
    if ($ldata){
        my @list = split /, /, $ldata;
        foreach my $med (@list){
            if ($med) {
                $med =~ s/[0-9]+.*$//;
                unless (exists($meds{$med})){
                    $meds{$med} = 1;
```

```
                    $patients{$gay_id}{$med} = 1;
                }else{
                    $meds{$med}++;
                    $patients{$gay_id}{$med} = 1;
                }
            }
        }
    }
}
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
foreach my $med (sort keys %meds){
    print ODF "$med $meds{$med}\n";
}

close ODF;
```



pues basicamente hay un total de 1872 palabras que se repiten mas de 10 veces, dejando un total de 14070 palabras que pueden considerarse mal escritas. Mal vamos.

¿Como lo hacemos?

Pero en principio esto no es malo ya que habra 1872 reglas. Voy a intentar calcular la distancia de cada palabra a estas palabras guia (las que mas se repiten) y agrupar cada una a la mas cercana. Si esta distancia no es suficientemente pequeña voy a suponer que la palabra no entra en ninguna regla y es un medicamento que se administra raras veces, o una palabra comun, o esta tan mal escrita que no sabria ni como empezar con ella.

La distancia entre palabras se calcula como la distancia de Levenshtein. Hay otras (he probado alguna pero la cosa no cambia mucho) pero con esta es bastante para lo que queremos.

Nota: La implementacion en "pure Perl" es muy lenta para calcular tantas distancias. Afortunadamente, existe una implementacion en C compilable y usable como modulo Perl que es mucho mas rapida.

Primero voy a separar las que se repiten mas de 10 veces,

```
$ awk {'if($2>10) print $1'} medrank.txt  > medbest.txt
$ head medbest.txt
aas
abilify
abril
ac
acabel
acalka
acarbosa
acetensil
acetil
acetilcisteina
```

OK, vamos a leer estas palabras y calcular la distancia de todas las palabra a estas, agrupamos en la de menor distancia. Para segurarnos, ponemos un *threshold*, si este no se supera, no agrupamos nada y mandamos lapalabra a otra lista. 🤔

Bah, basicamente leer, comparar y escribir,

rules.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use Data::Dump qw(dump);
use Text::Levenshtein::XS qw(distance);

my $dtresh = 3;
my $ikfile = "medbest.txt";
my $idfile = "medrank.txt";
my $ofile = "med_rules_l.txt";
my $obfile = "med_norules_l.txt";

my @medkw;
open IDF, "<$ikfile" or die "No such file\n";
while (<IDF>) {
    chomp;
    push @medkw, $_;
}
close IDF;

my @medw;
open IDF, "<$idfile" or die "No such file\n";
while (<IDF>) {
    chomp;
    (my $mname) = /^(\w*)\s+\d+$/;
    push @medw, $mname;
```

```
}
close IDF;

my %medgroups;
my @alonemeds;
foreach my $medword (@medw){
    my $mindist = 1000;
    my $keyguide = "";
    foreach my $medkey (@medkw){
        my $dist = distance($medword, $medkey, $dtresh);
        if ((defined $dist) && ($dist < $mindist)) {
            $mindist = $dist;
            $keyguide = $medkey;
        }
    }
    if ($keyguide) {
        push @{$medgroups{$keyguide}}, $medword;
        print "$medword --> $keyguide\n";
    }else{
        push @alonemeds, $medword;
        print "$medword --> no rules!\n";
    }
}

open ODF, ">$ofile" or die "Could not create file\n";
foreach my $mgroup (sort keys %medgroups){
    print ODF "$mgroup: ",join("|", sort @{$medgroups{$mgroup}}), "\n";
}
close ODF;
open ODF, ">$obfile" or die "Could not create file\n";
foreach my $med (@alonemeds){
    print ODF "$med\n";
}
close ODF;
```

El resultado es un archivo facilmente *parseable* con una lista de 1872 reglas,

```
> head med_rules_l.txt
aas:
aa|aaa|aas|aasl|abans|abasn|abse|achs|adias|ahy|aics|alas|als|ambas|anat|aqa
s|areas|as|bas|base|cajas|camas|cas|casa|casi|caso|dais|daxas|fase|gab|hrs|n
aas|paar|pase|paso|past|rash|vaso
abilify: abiliby|abilifi|abilify|abillify|abilyfi|abylifi
abril: abril|abrir|agil|nuril|pril|ril
ac:
ab|ac|ace|aci|acmd|acp|adic|ag|ahce|am|an|ao|ar|av|avc|ax|az|bach|back|canc|
faci|hac|irc|marc|nac|oxc|pc|puc|pvc|rc|saca|tc
acabel: acaba|acaban|acabar|acabel|acabo|acobel|acoxel|tasabel
acalka: acalka|acalkia|adalta|alcalka|calra
```

```
acarbosa: acabosa|acarbosa
acetensil: acentensil|acetensi|acetensil|actensil
acetil: acerl|acetil|acetina|actiq|aretdil|bicetil|retil
acetilcisteina:
acatilcisteina|aceticiteina|acetilcesteina|acetilcisteiana|acetilcisteina|ac
etilcisteinia|acetilcistena|acetilcisterina|acetilcistina|acetilsisteina|cet
ilcisteina
```

Evidentemente esta lista de reglas ha de revisarse.

Por otro lado, hay un total de 4411 palabras que el algoritmo no ha sido capaz de asociar a ninguna regla. En principio el resultado no es malo ya que la lista de columnas disminuiria a 6283 columnas, menos de la mitad. No obstante, despues de la edicion de las reglas esto podria subir a unas 7000 columnas (que tampoco es tan malo, creo).

El siguiente paso es ahora, editar el archivo de reglas. Las reglas pueden editarse, unirse, borrarse, mientras que se mantenga la estructura del archivo. El siguiente paso sera unificar todas las palabras que sigan estas reglas y las que no esten en ninguna regla tratarlas como variable independiente.

Nota: tambien podria hacerse un archivo con las palabras que no queremos incluir como variables. Ejemplo: *abandonado, abdominal, acceder, alrededor,agua*. que se yo.

Primera revision

Tenemos una desicion sobre la reglas correctas o incorrectas,

```
> head med_rules_ed_0219.csv
0;aas:
aa|aaa|aas|aasl|abans|abasn|abse|achs|adias|ahy|aics|alas|als|ambas|anat|aqa
s|areas|as|bas|base|cajas|camas|cas|casa|casi|caso|dais|daxas|fase|gab|hrs|n
aas|paar|pase|paso|past|rash|vaso
1;abilify: abiliby|abilifi|abilify|abillify|abilyfi|abylifi
0;abril: abril|abrir|agil|nuril|pril|ril
0;ac:
ab|ac|ace|aci|acmd|acp|adic|ag|ahce|am|an|ao|ar|av|avc|ax|az|bach|back|canc|
faci|hac|irc|marc|nac|oxc|pc|puc|pvc|rc|saca|tc
0;acabel: acaba|acaban|acabar|acabel|acabo|acobel|acoxel|tasabel
1;acalka: acalka|acalkia|adalta|alcalka|calra
1;acarbosa: acabosa|acarbosa
1;acetensil: acentensil|acetensi|acetensil|actensil
1;acetil: acerl|acetil|acetina|actiq|aretdil|bicetil|retil
1;acetilcisteina:
acatilcisteina|aceticiteina|acetilcesteina|acetilcisteiana|acetilcisteina|ac
etilcisteinia|acetilcistena|acetilcisterina|acetilcistina|acetilsisteina|cet
ilcisteina
```

vamos a escoger solo las correctas,

```
>  awk -F";" {'if($1) print $2'} med_rules_ed_0219.csv >
med_rules_ed_0219.txt
```

```
> head med_rules_ed_0219.txt
abilify: abiliby|abilifi|abilify|abillify|abilyfi|abylifi
acalka: acalka|acalkia|adalta|alcalka|calra
acarbosa: acabosa|acarbosa
acetensil: acentensil|acetensi|acetensil|actensil
acetil: acerl|acetil|acetina|actiq|aretdil|bicetil|retil
acetilcisteina:
acatilcisteina|aceticiteina|acetilcesteina|acetilcisteiana|acetilcisteina|ac
etilcisteinia|acetilcistena|acetilcisterina|acetilcistina|acetilsisteina|cet
ilcisteina
acetilsalicilico: acetilsalicilic|acetilsalicilico|acetilsalicitilico
acfol: acfgol|acfol|acfoll|acfrol|acofol|acpor|actol|actrol|affol
acido:
aacido|acdo|acid|acidez|acido|acidos|acifo|actino|activo|acude|adido|aidor|a
ixo|animo|arico|asio|ayudo|cacido|cido|leido
acovil: acnvis|acosia|acovil|acovul|acuvil|alcovil|movil
```

Todavia hay problemas que editar a mano,

```
alparazolam: alparazolam|alparzolam|alpprazolam|alprrazolam|laparazolam
alphagan: alfagan|alpagan|alpahgan|alphagan|alphagn|alphahgan|aphagan
alprazoalm:
alpraqzopalm|alprazlaom|alprazloam|alprazoal|alprazoalm|alprazolm|alprazolma
alprazolam:
aalprazolam|aklprazolam|alaprazolam|alkprazolam|aloprazolam|alpraqzolam|alpr
axolam|alprazalam|alprazaolam|alprazola|alprazolam|alprazolanm|alprazoloam|a
lprazolom|alprazoma|alprazpolam|alprazxolam|alprezolam|alprozolam|alprzolam|
alprzxolam|alrazolam|alrpazolam|aprazolam|laprazolam|liprazolam|loprazolam|l
prazolam|prazolam
alprazolan: alpraazolan|alprazolan|aprazolan
```

pero aqui no hay mas remedio que editar a mano y pegarlas.

```
alparazolam:
alparazolam|alparzolam|alpprazolam|alprrazolam|laparazolam|alpraqzopalm|alpr
azlaom|alprazloam|alprazoal|alprazoalm|alprazolm|alprazolma|aalprazolam|aklp
razolam|alaprazolam|alkprazolam|aloprazolam|alpraqzolam|alpraxolam|alprazala
m|alprazaolam|alprazola|alprazolam|alprazolanm|alprazoloam|alprazolom|alpraz
oma|alprazpolam|alprazxolam|alprezolam|alprozolam|alprzolam|alprzxolam|alraz
olam|alrpazolam|aprazolam|laprazolam|liprazolam|loprazolam|lprazolam|prazola
m|alpraazolan|alprazolan|aprazolan
```

Usando las reglas y construyendo la matriz

Una vez editadas las reglas, podemos ejecutar el mismo procedimiento pero ahora hay que comprobar cada palabra contra cada regla. Esto va a demorar un poco mas el procesamiento pero no creo que nada que no se pueda esperar.

A ver como es ahora la cosa,

parser2.pl

```
#!/usr/bin/perl

use strict;
use warnings;

my $ifile = "medica.csv";
my $ofile = "med2.csv";
my $rules_file = "med_rules_l.txt";

my %meds;
my %patients;
my %wrules;
# Leyendo las reglas
open RDF, "<$rules_file" or die "No rules file";
while (<RDF>){
    (my $rkey, my $rvalue) = /^(\w*): (.*)$/;
    $wrules{$rkey} = $rvalue unless !$rkey;
}
close RDF;
# Leyendo datos y comparando
open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
    (my $gay_id, my $ldata) = /^.*;(\d*)\.0;(.*)$/;
    if ($ldata){
        my @list = split /, /, $ldata;
        foreach my $med (@list){
            if ($med) {
                $med =~ s/[0-9]+.*$//;
                # aqui compruebo cada regla contra la palabra que he
leido
                foreach my $rkey (sort keys %wrules){
                    if ($med =~ /^($wrules{$rkey})$/){
                        $med = $rkey;
                        last;
                    }
                }
                $meds{$med} = 1;
                $patients{$gay_id}{$med} = 1;
            }
        }
    }
}
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno";
foreach my $med (sort keys %meds){
    print ODF ",$med";
}
```

```
print ODF "\n";
foreach my $patient (sort keys %patients){
    print ODF "$patient";
    foreach my $med (sort keys %meds){
        if(exists($patients{$patient}{$med})){
            print ODF ",1";
        }else{
            print ODF ",0";
        }
    }
    print ODF "\n";
}
close ODF;
```

y efectivamente la ejecucion es lenta,

```
$ time ./parser2.pl

real    72m24.613s
user    72m17.175s
sys     0m0.695s
```

pero el resultado es el esperado, la matriz ahora tiene *solo* 6284 columnas.

```
> head -n 1 med2.csv | sed 's/,/\n/g' | wc -l
6284
```

Y pesa bastante menos,

```
$ ls -lh med2.csv
-rw-rw---- 1 osotolongo osotolongo 164M Feb 14 13:46 med2.csv
```

¿Se podra hacer mas rapido?

Voy a intentar una cosa. Si en lugar de tratar la regla con *regex*s lo hago como *keys* de un *hash*, solo tendria que preguntar si cada elemento existe en lugar de hacer una comparacion por muchas palabras cada vez. No se si sera mas rapido pero merece la pena intentarlo.

Primero hay que leer un poco distinto,

```
# Leyendo las reglas
open RDF, "<$rules_file" or die "No rules file";
while (<RDF>){
    (my $rkey, my $rvalue) = /^(\w*): (.*)$/;
    chomp $rvalue;
    my @lpat = split /\|/, $rvalue;
    %{$wrules{$rkey}} = map {$_ => 1} @lpat unless !$rkey;
```

```
}
close RDF;
```

y la comparacion es un poco mas elemental ahora,

```
                foreach my $rkey (sort keys %wrules){
                    if (exists($wrules{$rkey}{$med})){
                        $med = $rkey;
                        last;
                    }
                }
```

El resto quedaría igual. Las operaciones son las mismas.

Aqui el código final

parser2a.pl

```
#!/usr/bin/perl

use strict;
use warnings;

my $ifile = "medica.csv";
my $ofile = "med2.csv";
my $rules_file = "med_rules_l.txt";

my %meds;
my %patients;
my %wrules;
# Leyendo las reglas
open RDF, "<$rules_file" or die "No rules file";
while (<RDF>){
    (my $rkey, my $rvalue) = /^(\w*): (.*)$/;
    chomp $rvalue;
    my @lpat = split /\|/, $rvalue;
    %{$wrules{$rkey}} = map {$_ => 1} @lpat unless !$rkey;
}
close RDF;
# Leyendo datos y comparando
open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
    (my $gay_id, my $ldata) = /^.*;(\d*)\.0;(.*)$/;
    if ($ldata){
        my @list = split /, /, $ldata;
        foreach my $med (@list){
            if ($med) {
                $med =~ s/[0-9]+.*$//;
                # aqui compruebo cada regla contra la palabra que he
leido
```

```
                foreach my $rkey (sort keys %wrules){
                    if (exists($wrules{$rkey}{$med})){
                        $med = $rkey;
                        last;
                    }
                }
                $meds{$med} = 1;
                $patients{$gay_id}{$med} = 1;
            }
        }
    }
}
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno";
foreach my $med (sort keys %meds){
    print ODF ",$med";
}
print ODF "\n";
foreach my $patient (sort keys %patients){
    print ODF "$patient";
    foreach my $med (sort keys %meds){
        if(exists($patients{$patient}{$med})){
            print ODF ",1";
        }else{
            print ODF ",0";
        }
    }
    print ODF "\n";
}
close ODF;
```

Ahora a probarlo,

```
$ time ./parser2a.pl

real    8m18.609s
user    8m17.526s
sys     0m0.352s

$ ls -lh med2.csv
-rw-rw---- 1 osotolongo osotolongo 164M Feb 15 10:37 med2.csv

$  head -n 1 med2.csv | sed 's/,/\n/g' | wc -l
6284
```

Eso fue rapido. Definitivamente nos quedamos con la segunda opcion.

Primera prueba real

```
osotolongo@daisy:~/Cloud/Documents/ACE/medicacion> wc -l
med_rules_ed_0219.txt
1222 med_rules_ed_0219.txt
osotolongo@daisy:~/Cloud/Documents/ACE/medicacion> scp -P 20022
med_rules_ed_0219.txt detritus.fundacioace.com:medicacion/
med_rules_ed_0219.txt
.............
[osotolongo@detritus medicacion]$ ./parser2a.pl
[osotolongo@detritus medicacion]$ ls -lh med2.csv
-rw-rw---- 1 osotolongo osotolongo 235M Feb 19 11:46 med2.csv
[osotolongo@detritus medicacion]$  head -n 1 med2.csv | sed 's/,/\n/g' | wc
-l
9022
```

V3

Vamos a considerar ahora que las reglas aprobadas son las unicas que corresponden a variables reales. Esto es todas las palabras pertenecientes a reglas no aprobadas o que no pertenecen a ninguna regla son descartadas.

La logica ahora cambia ligeramente pero afecta el resultado de manera importante.

```
        foreach my $med (@list){
            if ($med) {
                $med =~ s/[0-9]+.*$//;
                # aqui compruebo cada regla contra la palabra que he leido
                foreach my $rkey (sort keys %wrules){
                    if (exists($wrules{$rkey}{$med})){
                        $meds{$rkey} = 1;
                        $patients{$gay_id}{$rkey} = 1;
                        last;
                    }
                }
            }
        }
```

El resto es *pretty the same*. 😀

Vamos a retocar esto un poco

parser2b.pl

```
#!/usr/bin/perl

use strict;
use warnings;
```

```
use Data::Dump qw(dump);
use Mail::Sender;
use File::Basename qw(basename);

sub shit_done {
        my @adv = @_;

        my $sender = new Mail::Sender {smtp => 'localhost',
        from => "$ENV{'USER'}\@detritus.fundacioace.com"};

        $sender->MailMsg({to =>
"$ENV{'USER'}\@detritus.fundacioace.com",
        subject => 'Script terminado',
        msg => "$adv[0] ha terminado la ejecucion!\n\nRevise
$adv[1]\n"});
}



my $ifile = "medica.csv";
my $ofile = "med3.csv";
my $rules_file = "med_rules_ed_0219.txt";

my %meds;
my %patients;
my %wrules;
# Leyendo las reglas
open RDF, "<$rules_file" or die "No rules file";
while (<RDF>){
    (my $rkey, my $rvalue) = /^(\w*): (.*)$/;
    chomp $rvalue;
    my @lpat = split /\|/, $rvalue;
    %{$wrules{$rkey}} = map {$_ => 1} @lpat unless !$rkey;
}
close RDF;
# Leyendo datos y comparando
open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
    (my $gay_id, my $ldata) = /^.*;(\d*)\.0;(.*)$/;
    if ($ldata){
        my @list = split /, /, $ldata;
        foreach my $med (@list){
            if ($med) {
                $med =~ s/[0-9]+.*$//;
                # aqui compruebo cada regla contra la palabra que he
leido
                foreach my $rkey (sort keys %wrules){
                    if (exists($wrules{$rkey}{$med})){
                        $meds{$rkey} = 1;
                        $patients{$gay_id}{$rkey} = 1;
                        last;
```

```
                    }
                }
            }
        }
    }
}
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno";
foreach my $med (sort keys %meds){
    print ODF ",$med";
}
print ODF "\n";
foreach my $patient (sort keys %patients){
    print ODF "$patient";
    foreach my $med (sort keys %meds){
        if(exists($patients{$patient}{$med})){
            print ODF ",1";
        }else{
            print ODF ",0";
        }
    }
    print ODF "\n";
}
close ODF;

shit_done basename($ENV{_}), $ofile;
```

A probarlo,

```
osotolongo@daisy:~/Cloud/Documents/ACE/medicacion> scp -P 20022 parser2b.pl
detritus.fundacioace.com:medicacion/
parser2b.pl                                                   100%
1337    21.3KB/s   00:00
.......
[osotolongo@detritus medicacion]$ ./parser2b.pl
[osotolongo@detritus medicacion]$  head -n 1 med3.csv | sed 's/,/\n/g' | wc
-l
1223
[osotolongo@detritus medicacion]$ ls -lh med3.csv
-rw-rw---- 1 osotolongo osotolongo 32M Feb 19 22:12 med3.csv
```

El resultado es ahora un archivo mucho mas pequeño, con 1223 columnas. Este archivo solo contiene lascolumnas correspondientes a las palabras guia y agrupa solo las palabras contenidas en las reglas.

Seguimientos

Hasta ahora hemos considerado la manera de identificar la medicacion de los pacientes en la primera visita, esto es la medicacion que reciben al llegar al centro. EL objetivo siguiente es seguir como cambia la medicacion en las visitas sucesivas. Ahora, ademas del codigo interno del paciente, hemos de guardar el codigo que identifica la fecha de cada visita. De esta manera, han de guardarse multiples filas por cada paciente, permitiendo seguir el cambio de los medicamentos a lo largo del tiempo.

Afortunadamente, las reglas de edicion de palabras son las mismas. Con lo cual, tenemos adelantado la mitad del trabajo. Notese que en caso de surgir un medicamento nuevo, solo habria que buscar una *unica* regla para este medicamento y añadirla a las ya existentes. **Pensar como hacer esto**.

El archivo de entrada sigue el mismo formato que antes, pero ligeramente diferente. Esto no es problema.

```
[osotolongo@detritus medicacion]$ head medica_seg.csv
interno;fecha;medicacion
19960171;40268;amlodipino, axura, dacortin, escitalopram, exelon,
gabapentina, imurel, lorazepam, omeprazol, por, trombocitopenia
19960171;40631;axura, citalopram, cp, exelon, gabapetnina, prache
19960171;40942;alprazola, axura, citalopram, deprax, exelon, isperdal, m
19960171;41221;axura, deprax, distraneurine, el, parch, prometax, seroquel,
si, solo
19960171;41452;adiro, axura, de, deprax, distarneurine, eficaz, es, eusan,
muy, no, o, parhce, pero, prometax, quetipaina, rescta, seroquel
19960171;41821;acetilcisteina, axurs, derpax, exelon, gemfibrozilo,
neurontin, omeprazol, seroquel, termol, voltaren
19960201;38502;alprazolam, axura, exelon, lisinopril, neurontin, omeprazol,
orfidal, plavix, seroquel, sinvastatina
19960201;38768;al, alprazolam, aricept, dia, ebixa, exelon, fluoxetina,
losartan, ni, no, reminyl, tromalyt
19960201;38993;adiro, alprazolam, ameride, axura, citalopram, deprax,
emportal, enalaril, exelon, gabapentina, ideos1, optruma, tenormin
```

Para guardar la fecha basta hacer un *%HoHoH* y guardar cada linea independientemente,

```
# Leyendo datos y comparando
open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
        (my $gay_id, my $vdata, my $ldata) = /^(\d*);(\d*);(.*)$/;
        if ($ldata){
                my @list = split /, /, $ldata;
                foreach my $med (@list){
                        if ($med) {
                                $med =~ s/[0-9]+.*$//;
                                # aqui compruebo cada regla contra la
palabra que he leido
                                foreach my $rkey (sort keys %wrules){
                                        if (exists($wrules{$rkey}{$med})){
```

```
                                                $meds{$rkey} = 1;
$patients{$gay_id}{$vdata}{$rkey} = 1;
                                                last;
                                        }
                                }
                        }
                }
        }
}
close IDF;
```

Para escribirlo hay que desglosar el *%HoHoH* en cada paso. Se ordena primero por ID del paciente y luego por codigo de fecha,

```
open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno, Fecha";
foreach my $med (sort keys %meds){
        print ODF ", $med";
}
print ODF "\n";
foreach my $patient (sort keys %patients){
        for my $idate (sort keys %{$patients{$patient}}){
                print ODF "$patient, $idate";
                foreach my $med (sort keys %meds){
                        if(exists($patients{$patient}{$idate}{$med})){
                                print ODF ", 1";
                        }else{
                                print ODF ", 0";
                        }
                }
                print ODF "\n";
        }
}
close ODF;
```

Ponemos todo junto

parser3.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use Data::Dump qw(dump);
use Mail::Sender;
use File::Basename qw(basename);

sub shit_done {
        my @adv = @_;
```

```
        my $sender = new Mail::Sender {smtp => 'localhost',
        from => "$ENV{'USER'}\@detritus.fundacioace.com"};

        $sender->MailMsg({to =>
"$ENV{'USER'}\@detritus.fundacioace.com",
        subject => 'Script terminado',
        msg => "$adv[0] ha terminado la ejecucion!\n\nRevise
$adv[1]\n"});
}



my $ifile = "medica_seg.csv";
my $ofile = "med4.csv";
my $rules_file = "med_rules_ed_0219.txt";

my %meds;
my %patients;
my %wrules;
# Leyendo las reglas
open RDF, "<$rules_file" or die "No rules file";
while (<RDF>){
        (my $rkey, my $rvalue) = /^(\w*): (.*)$/;
        chomp $rvalue;
        my @lpat = split /\|/, $rvalue;
        %{$wrules{$rkey}} = map {$_ => 1} @lpat unless !$rkey;
}
close RDF;
# Leyendo datos y comparando
open IDF, "<$ifile" or die "No such file";
while (<IDF>) {
        (my $gay_id, my $vdata, my $ldata) = /^(\d*);(\d*);(.*)$/;
        if ($ldata){
                my @list = split /, /, $ldata;
                foreach my $med (@list){
                        if ($med) {
                                $med =~ s/[0-9]+.*$//;
                                # aqui compruebo cada regla contra la
palabra que he leido
                                foreach my $rkey (sort keys %wrules){
                                        if
(exists($wrules{$rkey}{$med})){
                                                $meds{$rkey} = 1;
$patients{$gay_id}{$vdata}{$rkey} = 1;
                                                last;
                                        }
                                }
                        }
                }
        }
}
```

```
close IDF;

open ODF, ">$ofile" or die "Could not open file!!!";
print ODF "Interno, Fecha";
foreach my $med (sort keys %meds){
        print ODF ", $med";
}
print ODF "\n";
foreach my $patient (sort keys %patients){
        for my $idate (sort keys %{$patients{$patient}}){
                print ODF "$patient, $idate";
                foreach my $med (sort keys %meds){
                        if(exists($patients{$patient}{$idate}{$med})){
                                print ODF ", 1";
                        }else{
                                print ODF ", 0";
                        }
                }
                print ODF "\n";
        }
}
close ODF;

shit_done basename($ENV{_}), $ofile;
```

y a correr,

```
[osotolongo@detritus medicacion]$ time ./parser3.pl

real    5m15.635s
user    5m14.644s
sys 0m0.450s

[osotolongo@detritus medicacion]$ ls -lh med4.csv
-rw-rw-r-- 1 osotolongo osotolongo 192M Mar 27 10:21 med4.csv

[osotolongo@detritus medicacion]$ awk -F"," '{print $1, $2}' med4.csv | head
Interno  Fecha
19960171  40268
19960171  40631
19960171  40942
19960171  41221
19960171  41452
19960171  41821
19960201  38502
19960201  38768
19960201  38993

[osotolongo@detritus medicacion]$ head -n 1 med4.csv | sed 's/,/\n/g' | wc -
```

```
l
1224

[osotolongo@detritus medicacion]$ wc -l med4.csv
54406 med4.csv
```

y ahi queda un matriz de 1224 columnas y 54405 filas, ordenada segun el interno del paciente y el codigo de fecha de la visita. 😎

Revisando las reglas

Al introducir una nueva base de datos existe la posibilidad de que se hayan introducido medicamentos que no existen en las reglas previas. Ha de revisarse la lista de reglas para esta nueva base de datos.

Cambiamos los archivos de entrada y salida,

```
my $ifile = "medica_seg.csv";
my $ofile = "medrank_seg.csv";
```

y buscamos el *rank* nuevo,

```
$ ./rank.pl
$ awk {'if($2>10) print $1'} medrank_seg.csv  > medbest_seg.txt
$ head medbest_seg.txt
a
aas
abilify
abril
ac
acabel
acalka
acarbosa
acetensil
acetil
```

Lo mismo para las reglas,

```
my $ikfile = "medbest_seg.txt";
my $idfile = "medrank_seg.csv";
my $ofile = "med_rules_ld_seg.txt";
my $obfile = "med_norules_ld_seg.txt";
```

```
$ ./rules2.pl
$ head med_rules_ld_seg.txt
a:
a|aa|ab|ag|am|an|ao|ar|as|av|ax|az|ea|f|ga|gab|ia|iba|ja|ma|na|oma|pa|sa|t|x
a
aas:
```

```
aaa|aas|aasl|abans|abasn|abse|achs|adias|alas|als|ambas|anat|aqas|areas|asa|
asma|bas|base|cajas|camas|cas|casa|casi|caso|dais|daxas|fase|hrs|naas|pase|p
aso|past|rash|vaso
abilify: abiliby|abilifi|abilify|abillify|abilyfi|abylifi
abril: abril|abrir|agil|nuril|pril|ril
ac:
ac|ace|aci|acmd|acp|adic|avc|bach|back|canc|faci|hac|irc|marc|nac|oxc|pc|puc
|pvc|rc|saca|tc
acabel: acaba|acaban|acabar|acabel|acabo|acobel|acoxel|tasabel
acalka: acalka|acalkia|aclara|alcalka|calra
acarbosa: acabosa|acarbosa
acetensil: acentensil|acetensi|acetensil|actensil
acetil: aceite|acetil|acetina|actiq|bicetil|retil
```

Tomemos al archivo de reglas editadas,

```
$ head med_rules_ed_0219.csv
0;aas:
aa|aaa|aas|aasl|abans|abasn|abse|achs|adias|ahy|aics|alas|als|ambas|anat|aqa
s|areas|as|bas|base|cajas|camas|cas|casa|casi|caso|dais|daxas|fase|gab|hrs|n
aas|paar|pase|paso|past|rash|vaso
1;abilify: abiliby|abilifi|abilify|abillify|abilyfi|abylifi
0;abril: abril|abrir|agil|nuril|pril|ril
0;ac:
ab|ac|ace|aci|acmd|acp|adic|ag|ahce|am|an|ao|ar|av|avc|ax|az|bach|back|canc|
faci|hac|irc|marc|nac|oxc|pc|puc|pvc|rc|saca|tc
0;acabel: acaba|acaban|acabar|acabel|acabo|acobel|acoxel|tasabel
1;acalka: acalka|acalkia|adalta|alcalka|calra
1;acarbosa: acabosa|acarbosa
1;acetensil: acentensil|acetensi|acetensil|actensil
1;acetil: acerl|acetil|acetina|actiq|aretdil|bicetil|retil
1;acetilcisteina:
acatilcisteina|aceticiteina|acetilcesteina|acetilcisteiana|acetilcisteina|ac
etilcisteinia|acetilcistena|acetilcisterina|acetilcistina|acetilsisteina|cet
ilcisteina
```

Tengo una lista de reglas en *med_rules_ed_0219.csv*. Esta lista he de eliminarla de *med_rules_ld_seg.txt*.

Saco los headers de la primera lista,

```
$ awk -F":" '{print $1}' med_rules_ed_0219.csv | awk -F";" '{print $2}' >
list_headers.txt
```

Ahora, estos headers hay que tomarlos y quitar de la nueva base de reglas, todas las reglas que ya existan. Voy hacermeun programillapara entenderlo,

quita.pl

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

```
use strict;
use warnings;
use Data::Dump qw(dump);

my $mhfile = "list_headers.txt";
my @mhead;
open IDF, "<$mhfile" or die "There is no list file\n";
@mhead = <IDF>;
close IDF;
chomp @mhead;

my $rfile = "med_rules_ld_seg.txt";
my %medrules;
open IDF, "<$rfile" or die "There is no rules file\n";
while (<IDF>) {
    if(/\w+:.*/) {
        (my $hrule) = /(\w+):.*/;
        $medrules{$hrule} = $_;
    }
}
close IDF;
foreach my $mh (@mhead){
    if(exists($medrules{$mh})){ delete($medrules{$mh})};
}
foreach my $queda (sort keys %medrules){
    print "$medrules{$queda}";
}
```

A ver que pasa,

```
$ ./quita.pl
a:
a|aa|ab|ag|am|an|ao|ar|as|av|ax|az|ea|f|ga|gab|ia|iba|ja|ma|na|oma|pa|sa|t|x
a
actual: actual|acutal|aigual|anual
actualmente: actualemnt|actualment|actualmente|actuamente|acutlamente
adicional: addicional|adiccional|adicioanl|adicionado|adicional|adiconal
admon: admon|admosn|limon
agitacion: agitaciion|agitacion|agitaicon|agitcion|agoitacion
al: al|ala|alli|alp|alt|dl|enal|kal|lal|oal|ol|olm|real|rl|sal|tal|wl|yl
algo: alga|algo|alto|angor|kleo|lago|ulco
algun: agut|alfuon|algin|algun|alguno|alken|almus|alquen
alguna: alguna|algunas|altana|angina
```

Y dado que todo lo que me queda es porqueria, la base de datos de reglas anterior **es válida**.

Medicacion antes de la primera visita

Para sacar la medicacion antes de la primera visita tenemos una nueva base de datos.

```
[osotolongo@detritus medicacion]$ head medicacion_anam.csv
interno;fecha;medicacion
19960003.0;35093.0;tensoprem, seguril, plurimen, masdil, k, boi, aremis
19960004.0;35065.0;nerdipina, meleril, distraneurine, cisordimol
19960005.0;35073.0;
19960006.0;35081.0;si, s, remontal, meleril, m, hidroferol, escazine,
ciclofalina, aneurol, al
19960007.0;35493.0;terma, si, sandoz, rmula, presa, no, nimodipino,
nicergolina, mesos, magistral, la, i, hidroferol, gin, f, eskazine, durant,
disgren, disfosfen, difosfen, dies, dia, descansar, de, codeina, citicolina,
choque, calcium, cal, ampolles, al, a
19960008.0;35093.0;
19960009.0;35080.0;largactil, intramuscular, hores, eskazine, en, d, cas,
agitaci
19960010.0;35121.0;becozyme
19960012.0;35082.0;
```

Sacamos primero el *rank* de las palabras y seleccionamos las palabras que se repiten mas de 10 veces,

```
[osotolongo@detritus medicacion]$ ./rank.pl
[osotolongo@detritus medicacion]$ head medrank_anam.csv
_ 2
a 577
aa 2
aaaadiro 1
aas 1259
ab 2
abacavir 1
abafranil 1
abamed 1
abans 6
[osotolongo@detritus medicacion]$ awk {'if($2>10) print $1'}
medrank_anam.csv  > medbest_anam.txt
[osotolongo@detritus medicacion]$ head medbest_anam.txt
a
aas
abilify
ac
accuhaler
aceclofenaco
acetensil
acetilcisteina
acfol
acido
```

Editamos el archivo de crear las reglas para manejar archivos nuevos,

```
my $ikfile = "medbest_anam.txt";
my $idfile = "medrank_anam.csv";
my $ofile = "med_rules_ld_anam.txt";
my $obfile = "med_norules_ld_anam.txt";
```

Y tenemos un nuevo archivo de reglas,

```
[osotolongo@detritus medicacion]$ ./rules3.pl
[osotolongo@detritus medicacion]$ head med_rules_ld_anam.txt
a: |_|a|aa|ab|ad|am|an|ao|ap|ar|as|axe|da|ia|ma|na|pa|pau|ra|t|ta|tab
aas:
aas|abans|acad|als|altas|ams|anal|aos|asa|asma|bajas|base|cas|casi|cis|daxas
|eas|faes|fase|gafas|gasas|hrs|iacs|idas|nias|paso|vas|was
abilify: abiilifi|abiliby|abilifi|abilify|abillity|abilyfi
ac: ac|aco|avc|bac|bach|dmc|foc|hac|ic|kcl|marc|mc|pic|rec|ric|tec|tic
accuhaler: acccuhale|accuhaler|acuhaler
aceclofenaco: aceclofenaco
acetensil: acentensil|acentesil|acetensil|atentensil
acetilcisteina: acetilcisteina|acetilsisteina|aetilcisteina
acfol: acfl|acfol|actol|afol|alfol
acido:
acdo|acid|acido|acidos|aciso|acude|animo|arido|axigo|caida|cid|cids|incido|l
cico|marido|sacado|sido
```

Editamos el script de comparacion,

```
my $rfile = "med_rules_ld_anam.txt";
```

y obtenemos una nueva lista de reglas que no estan presentes en la DB primaria,

```
[osotolongo@detritus medicacion]$ head diff_anam.txt
a: |_|a|aa|ab|ad|am|an|ao|ap|ar|as|axe|da|ia|ma|na|pa|pau|ra|t|ta|tab
accuhaler: acccuhale|accuhaler|acuhaler
aceclofenaco: aceclofenaco
actual: actual|actuales|altal|anual|ictal
actualmente: actualemnt|actualment|actualmente|acutalmente|atualment
acuretic: acuaretic|acuretic
adenuric: adenuric
admon: admon|admosn|animon
al:
al|ala|ald|alf|alfa|alka|alta|apli|cl|fal|glp|il|mal|mala|nrl|ol|plu|pul|rea
l|ull|vial|xazl|yl
algo: algo|algoe|algun|alto|llego|palto|salvo|ulco
```

Esta lista ha de editarse a mano,

```
[osotolongo@detritus medicacion]$ head rules_anam_add.txt
accuhaler: acccuhale|accuhaler|acuhaler
```

```
aceclofenaco: aceclofenaco
acuretic: acuaretic|acuretic
adenuric: adenuric
almogran: almogran
amitriptilina: amitriptilina|anuitriptilina
analgilasa: anagilasa|analgilasa
ansium: ansium|anxium|sandium
asasantin: asasantin
atenolo: astenolo|atenol|atenolo
```

Ahora se añaden estas reglas a las anteriores,

```
[osotolongo@detritus medicacion]$ cat med_rules_ed_0219.txt
rules_anam_add.txt > med_rules_ed_0219_plus.txt
```

y sacamos la matriz con estas nuevas reglas.

Lo he hecho de las dos maneras, escribiendo las fechas y sin tenerlas en cuenta,

```
[osotolongo@detritus medicacion]$ ls parser4*
parser4a.pl  parser4.pl
[osotolongo@detritus medicacion]$ ls -lh med5*
-rw-rw---- 1 osotolongo osotolongo 73M Apr  5 21:34 med5a.csv
-rw-rw---- 1 osotolongo osotolongo 72M Apr  5 21:32 med5.csv
[osotolongo@detritus medicacion]$ awk -F"," '{print $1, $2}' med5a.csv |
head
Interno  Fecha
19960003  1996/01/31
19960004  1996/01/03
19960006  1996/01/19
19960007  1997/03/06
19960009  1996/01/18
19960010  1996/02/28
19960016  1996/01/24
19960017  1996/01/24
19960021  1995/12/30
[osotolongo@detritus medicacion]$ wc -l med5*
    19967 med5a.csv
    19868 med5.csv
    39835 total
[osotolongo@detritus medicacion]$ head -n 1 med5.csv | sed 's/,/\n/g' | wc -
l
1260
[osotolongo@detritus medicacion]$ head -n 1 med5a.csv | sed 's/,/\n/g' | wc
-l
1261
```

Las matrices difieren ligeramente, por algun registro repetido, que en el primer caso se acumulan y en el segundo se separan. Ejemplo,

```
19960098  1996/04/11
```

```
19960098  2017/02/04
...
19960171  1996/06/08
19960171  2009/04/08
...
19960235  1996/10/12
19960235  1998/10/16
...
```

From:
https://mail.fundacioace.com/wiki/ - **Detritus Wiki**

Permanent link:
https://mail.fundacioace.com/wiki/doku.php?id=medicacion

Last update: **2020/08/04 10:58**