Tuning Algorithm::Diff

Autor: Helmut Wollmersdorfer

CPAN, github: wollmers

Deutscher Perl-Workshop 2016

https://github.com/wollmers/talks-gpw2016

Problem

- Vergleich zweier Sequenzen
- Ausrichtung mit maximaler Ähnlichkeit
- Ergebnis als Positionen (Flexibilität)
- Sequenzen als Arrays of Strings
- Elemente: Chars, Wörter, Zeilen etc.

Ausrichtung

```
Chrerrplzon
Choerephon
```

0	**1**	2	**3**	**4**	5	**6**	7	8	**9**	**10**
C	**h**	r	**e**	**r**	r	**p**	l	z	**o**	**n**
|	|		|	|		|			|	|
C	**h**	o	**e**	**r**	e	**p**	h		**o**	**n**
0	**1**	2	**3**	**4**	5	**6**	7		**8**	**9**

```
0 1 3 4 6 9 10
0 1 3 4 6 8  9
```

Vorteil durch Beschränkung

- Alt: `LCSidx(\@s1,\@s2,\&hash,\&cmp)`
- Neu: `LCS(\@s1,\@s2)`

Ersparnis:

- ~35 LoCs
- Prüfen und Calls der Code-Refs (langsam)

Twiddling, Squeezing

```
# OLD
sub _withPositionsOfInInterval{
    my $aCollection = shift;    # array ref
    my $start       = shift;
    my $end         = shift;
    my $keyGen      = shift;
    my %d;
    my $index;
    for ($index = $start ; $index <= $end ; $index++) {
        my $element = $aCollection->[$index];
        my $key = &$keyGen( $element, @_ );
        if ( exists( $d{$key} ) ) {
            unshift ( @{ $d{$key} }, $index );
        }
        else {
            $d{$key} = [$index];
        }
    }
    return wantarray ? %d : \%d;
}
```

```
# NEW
my $bMatches;
unshift @{ $bMatches->{$b->[$_]} },$_ for $bmin..$bmax;
```

Inlining

- Alt: 4 subs
- Neu: 1 sub

Vorteil:

- Weniger Zeilen
- Weniger Parameterübergaben
- Weniger Kontext erzeugen und aufräumen

Komplexität

- Alt: 37 Punkte McCabe, 162 LoC
- Neu: 32 Punkte McCabe,   55 LoC

Benchmark

```
              Rate LCSidx LCSnew  LCSXS S::Sim
LCSidx   25025/s     --   -39%   -55%   -98%
LCSnew   41152/s    64%     --   -25%   -96%
LCSXS    55188/s   121%    34%     --   -95%
S::Sim 1086957/s  4243%  2541%  1870%     —
```

```
LCSidx  Algorithm::Diff::LCSidx()
LCSnew  LCS::Tiny::LCS()
LCSXS   Algorithm::Diff::XS::LCSidx()
S::Sim  String::Similarity::similarity()
```

Algorithms

Hunt-Szymanski 1977 (needs Hash)

- Algorithm::Diff(::XS)
- LCS::Tiny

Ukkonen 1985 / Myers 1986 (diagonals, recursive)

- String::Similarity (C/XS from Gnu-Diff)
- LCS::XS (beta quality, not tuned)

not Perl-friendly

Bit Vectors

Hunt-Szymanski 1977 (Hash-Array-Integer)

'anna' → 'a' => [4,1]

Allison-Dix 1986 (Hash-[Array]-Word)

'anna' → 'a' => \b1001

Hyyroe 2004 (faster, LLCS, SES, Damerau)

- LCS::BV, P6: LCS-BV
- c-lcs-bv (github, in progress)

LCS::BV

Case: Chrerrplzon <> Choerephon

```
                       Rate
LCS                  6636/s
Algorithm::Diff     25599/s
LCS::Tiny           41353/s
Algorithm::Diff::XS 55351/s
LCS::BV             56888/s
```

Finished?

Tim Bunce:

STOP

HERE!

LCS::XS

The XS Bottleneck

```
                Rate Input    Output
A::D::XS      55351/s Arrays   Array
LCS::XSa     150905/s Arrays   Array
LCS::XSs     217375/s Strings Array
cLCS::XSs    238601/s Strings RLE-Array
S::Sim      1087949/s Strings Scalar (Rat)
```

c-lcs-bv (LLCS)

• Bob Jenkins Hash	~250 kHz
• Kernighan-Ritchie	~500 kHz
• Serial Map, 3 allocs	~2 Mhz
• 1 calloc	~4 MHz
• UTF-8	~1.7 Mhz
• VLA (stack „alloc“)	~7.5 Mhz
• VLA UTF-8	~2.3 MHz

Why?

- Ofun
- Train the brain
- Improve knowledge by challange
- Don't STOP here!
- Need for speed

Next Steps

- Algorithm::Diff::Tiny (BV, reduced API)
- Algorithm::Diff::Formats
- LCS::XS (make it rock solid)
- LCS::Similar::XS
- Star centered multi-align in C

Questions?

???