levenshtein egg

Levenshtein edit distance
Extension for Chicken Scheme
Version 1.3

Kon Lovett

Table of Contents

1 About this egg

1.1 Version history

`1.3`	Major changes
`1.2`	Switched to array-lib
`1.1`	Requirement for srfi-43 was wrong [Thanks to Benedikt Rosenau]
`1.0`	Initial release

1.2 Requirements

This egg requires the following extensions:

`utf8`, `numbers`, `procedure-surface`, `miscmacros`, `misc-extn`, `syntax-case`, `vector-lib`, `array-lib`

2 Documentation

Levenshtein is a collection of procedures providing various forms of the Levenshtein edit distance calculation.

The Levenshtein edit distance has been used for areas as diverse as soil sample and language dialect analysis. Not just for text strings.

2.1 8-bit Characters Only

Performs edit distance calculation for byte strings. All return the total edit cost.

```
(require-extension levenshtein-byte)
```

`levenshtein-distance/byte` [procedure]

```
(levenshtein-distance/byte SOURCE TARGET)
```

Calculates the edit distance from the `SOURCE` to the `TARGET`. All costs are unitary.

```
(require-extension levenshtein-transpose-byte)
```

`levenshtein-distance/transpose-byte` [procedure]

```
(levenshtein-distance/transpose-byte SOURCE TARGET)
```

Calculates the edit distance from the `SOURCE` to the `TARGET`, taking into account the Transpose operation. All costs are unitary.

2.2 Procedure Means

Supplies the arithmetic and string operations for the distance algorithms below.

Should you wish to supply your own 'means' please see the procedure-surface egg documentation, "levenshtein-*-surface.scm", and "levenshtein-*-means.scm" in this egg for more information.

```
(require-extension levenshtein-utf8-means)
```

`levenshtein-utf8-means` [procedure-means]

```
(levenshtein-utf8-means (levenshtein-string-surface))
```

Uses procedures from the utf8 egg.

```
(require-extension levenshtein-octet-means)
```

`levenshtein-octet-means` [procedure-means]

```
(levenshtein-octet-means (levenshtein-string-surface))
```

Uses procedures from SRFI-13.

```
(require-extension levenshtein-numbers-means)
```

`levenshtein-numbers-means` [procedure-means]

```
(levenshtein-numbers-means (levenshtein-numeric-surface))
```

Uses procedures from the numbers egg.

```
(require-extension levenshtein-gennum-means)
```

`levenshtein-gennum-means` [procedure-means]

`(levenshtein-gennum-means (levenshtein-numeric-surface))`

Uses only fixnum and flonum procedures.

`(require-extension levenshtein-fixnum-means)`

`levenshtein-fixnum-means` [procedure-means]

`(levenshtein-fixnum-means (levenshtein-numeric-surface))`

Uses only fixnum procedures.

2.3 Generic String

Performs edit distance calculation for strings.

`(require-extension levenshtein-generic-string)`

`levenshtein-distance/generic-string` [procedure]

`(levenshtein-distance/generic-string SOURCE TARGET [#:insert-cost 1] [#:delete-co`

Calculates the edit distance from the `SOURCE` to the `TARGET`.

`SOURCE` A string.

`TARGET` A string.

`insert-cost:`
A number. The edit cost of an insert.

`delete-cost:`
A number. The edit cost of a delete.

`substitute-cost:`
A number. The edit cost of a substitute.

`=?:` A procedure; (-> object object boolean). The equality predicate. Probably not useful to override the default predicate.

`limit-cost:`
A number or #f. Limit edit distance calculation to cost. Number type must be compatible with the numeric-means.

`numeric-means:`
A procedure-means. The arithmetic means.

`string-means:`
A procedure-means. The string means.

2.4 Generic Sequence

`(require-extension levenshtein-generic-sequence)`

`levenshtein-distance/generic-sequence` [procedure]

`(levenshtein-distance/generic-sequence SOURCE TARGET [#:insert-cost 1] [#:delete-`

Calculates the edit distance from the `SOURCE` to the `TARGET`.

`SOURCE` A vector, string, or list.

`TARGET` A vector, string, or list.

`insert-cost:`
: A number. The edit cost of an insert.

`delete-cost:`
: A number. The edit cost of a delete.

`substitute-cost:`
: A number. The edit cost of a substitute.

`get-work-vector:`
: A procedure; (-> number vector). Returns a work vector of length 'number', or larger.

`=?:` A procedure; (-> object object boolean). The equality predicate. Must handle the types of the source & target elements in either argument position!

`limit-cost:`
: A number or #f. Limit edit distance calculation to cost. Number type must be compatible with the numeric-means.

`numeric-means:`
: A procedure-means. The arithmetic means.

`string-means:`
: A procedure-means. The string means. Only used when source & target are strings.

The conversion from string to vector is dependent on the binding of 'string->list' at the time of the call to 'levenshtein-distance/generic-sequence. Can be an issue when argument types are mixed; string and vector, or string and list. String and string are passed on to 'levenshtein-distance/generic-string'

2.5 Only Vector - Baroque & Slow

Performs edit distance calculation for vectors. Allows definition of new edit operations. Will keep track of edit operations performed.

```
(require-extension levenshtein-vector)
```

`levenshtein-distance/vector*` [procedure]

```
(levenshtein-distance/vector* SOURCE TARGET [EDIT-OPER ...] [#:=? char=?] [#:oper
```

Calculates the edit distance from the source vector `SOURCE` to the target vector `TARGET`. Returns the total edit cost or (values <total edit cost> <performed operations matrix>).

`SOURCE` A vector.

`TARGET` A vector.

`EDIT-OPER`
: Edit operation definitions to apply. Defaults are the basic Insert, Delete, and Substitute.

`=?:` A procedure; (-> object object boolean). The equality predicate.

`operations:`
A boolean. Include the matrix of edit operations performed?

`numeric-means:`
A procedure-means. The arithmetic means.

`(require-extension levenshtein-path-iterator)`

`levenshtein-path-iterator` [procedure]

```
(levenshtein-path-iterator MATRIX)
```

Creates an optimal edit distance operation path iterator over the performed operations matrix `MATRIX`. The matrix is usually the result of an invocation of '(levenshtein-distance/vector* ... operations: #t)'.

Each invocation of the iterator will generate a list of the form: ((cost source-index target-index levenshtein-operator) ...). The last invocation will return #f.

2.6 Edit operations

Edit operation specification. A set of base operations is predefined, but may be overridden. The base set is identified by the keys Insert, Delete, Substitute, and Transpose. A printer and reader are provided for edit operations.

`(require-extension levenshtein-operators)`

`levenshtein-operator` [record]

`key`	A symbol. Key for the operation.
`name`	A string. Describes the operation.
`cost`	A number. The cost of the operation.
`above`	A non-negative fixnum. How far back in the source.
`left`	A non-negative fixnum. How far back in the target

`make-levenshtein-operator` [procedure]

```
(make-levenshtein-operator KEY NAME COST ABOVE LEFT)
```

Returns a new edit operator.

`levenshtein-operator?` [procedure]

```
(levenshtein-operator? OBJECT)
```

Is the `OBJECT` a levenshtein operator?

`clone-levenshtein-operator` [procedure]

```
(clone-levenshtein-operator EDIT-OPERATION [#:key] [#:name] [#:cost] [#:above] [#
```

Returns a duplicate of the `EDIT-OPERATION`, with field values provided by the optional keyword arguments. EDIT-OPERATION may be the key of the already defined edit operation.

`levenshtein-operator-ref` [procedure]

```
(levenshtein-operator-ref KEY)
```

Get the definition of an edit operation.

`levenshtein-operator-set!` [procedure]

`(levenshtein-operator-set! EDIT-OPERATION)`

Define an edit operation.

`levenshtein-operator-delete!` [procedure]

`(levenshtein-operator-delete! EDIT-OPERATION)`

Removes the `EDIT-OPERATION` definition. EDIT-OPERATION may be the key of the already defined edit operation.

`levenshtein-operator-reset` [procedure]

`(levenshtein-operator-reset)`

Restore defined edit operations to the base set.

`levenshtein-operator-equal?` [procedure]

`(levenshtein-operator-equal? A B)`

Are the levenshtein operators `A` & `B` equal for all fields?

2.7 Version 1.2 Compatibility

`(require-extension levenshtein)`

Warning - The numbers and utf8 eggs will be used!

`levenshtein-distance/string` [macro]

`(levenshtein-distance/string SOURCE TARGET [EQ INSERT-COST DELETE-COST SUBSTITUTE`

Calculates the edit distance from the source string `SOURCE` to the target string `TARGET`. The default equivalence procedure `EQ` is 'char=?'. The default costs `INSERT-COST`, `DELETE-COST`, and `SUBSTITUTE-COST` are unitary.

`levenshtein-distance/list` [macro]

`(levenshtein-distance/list SOURCE TARGET [EQ INSERT-COST DELETE-COST SUBSTITUTE-C`

Calculates the edit distance from the source list `SOURCE` to the target list `TARGET`. The default equivalence procedure `EQ` is 'equal?'. The default costs `INSERT-COST`, `DELETE-COST`, and `SUBSTITUTE-COST` are unitary.

`levenshtein-distance/vector` [macro]

`(levenshtein-distance/vector SOURCE TARGET [EQ INSERT-COST DELETE-COST SUBSTITUTE`

Calculates the edit distance from the source vector `SOURCE` to the target vector `TARGET`. The default equivalence procedure `EQ` is 'equal?'. The default costs `INSERT-COST`, `DELETE-COST`, and `SUBSTITUTE-COST` are unitary.

`levenshtein-distance/sequence` [macro]

`(levenshtein-distance/sequence SOURCE TARGET [EQ INSERT-COST DELETE-COST SUBSTITU`

Calculates the edit distance from the source sequence `SOURCE` to the target sequence `TARGET`. The default equivalence procedure `EQ` is 'char=?'. The default costs `INSERT-COST`, `DELETE-COST`, and `SUBSTITUTE-COST` are unitary.

`levenshtein-distance/scratch` [macro]

`(levenshtein-distance/scratch SOURCE TARGET [GET-SCRATCH EQ INSERT-COST DELETE-CO`

Calculates the edit distance from the source vector `SOURCE` to the target vector `TARGET`. The default equivalence procedure `EQ` is 'equal?'. The default costs `INSERT-COST`, `DELETE-COST`, and `SUBSTITUTE-COST` are unitary. The default get-scratch procedure `GET-SCRATCH` is make-vector.

Few, if any, programs will use this procedure directly. This is like levenshtein-distance/vector, but allows get-scratch to be specified. get-scratch is a procedure of one term, n, that yields a vector of length n or greater, which is used for record-keeping during execution of the Levenshtein algorithm. make-vector can be used for get-scratch, although some programs comparing a large size or quantity of vectors may wish to reuse a record-keeping vector, rather than each time allocating a new one that will need to be garbage-collected.

3 Examples

```
(use levenshtein-vector levenshtein-path-iterator)

(define iter
        (levenshtein-path-iterator
                (levenshtein-distance/vector* "YWCQPGK" "LAWYQQKPGKA" operations: #t))
; ignoring interpreter feedback
(define r0 (iter))
(define t0 r0)
(define r1 (iter))
(define r2 (iter))
(define r3 (iter))
(define r4 (iter))
(define r5 (iter))
(iter)
; r0 now has #f, since the iterator finishes by returning to the initial caller, which is t
; body of '(define r0 (iter))', thus re-binding r0. However, t0 has the original returned v

; Please see the 'levenshtein*test.scm' files in the levenshtein egg for more examples.
```

4 References

Pictures of Pr. Levenstein

- Levenshtein distance @ Wikipedia
- Levenshtein Distance
- Levenshtein distance
- Talk:Levenshtein distance @ Wikipedia

5 License

Index