NAME

aregex - approximate (fuzzy) string matching with regular expressions

SYNOPSIS

```
@load "aregex"
```

success = amatch(str, regex [, cost|costs [, submatches]])

DESCRIPTION

The TRE library (ref. below) provides approximate matching regex capabilities. A match between two strings that differ in some number of characters will be found when the cost of character insertions, deletions and substitutions does not exceed some specified maximum cost. For example,

```
"abcdef"
"abcxdef"  # one insertion
"abdef"    # one deletion
"abxdef"   # one substitution
```

The cost of the match (the Levenshtein distance between strings) can be reported. This Gawk extension provides an interface with the *tre_regaexec()* function in the TRE library, permitting the setting of all possible parameters for that function, and returning all possible information about a match.

Function summary

A single function, **amatch()** is provided, modeled on the Gawk *match()* function:

amatch(*str* **,** *regex* **[,** *cost*|*costs* **[,** *submatches* **]])**

This function takes two mandatory string arguments, and two optional arguments. *regex* is an **extended** regular expression (or plain string) to be matched against string *str*. Note that the regular expression *regex* is bounded by double-quotes, not by the usual Gawk slashes.

Setting approximate match costs

With only two arguments, the default maximum cost for the approximate match is set to 5 (and other costs are set as below). The maximum cost may also be set by the user via an optional **third argument**: either an integer (*cost*), or a member of a one-dimensional array (*costs*) indexed by `"max_cost"`. Setting maximum cost to 0 forces an exact regular expression match, as with Gawk's *match()*. Other members of the *costs* array with appropriate index values will set the parameters of the *regaparams_t* structure used by *tre_regaexec()*:

Array index	Parameter	Def val	Meaning
"cost_ins"	.cost_ins	1	Cost of one insertion
"cost_del"	.cost_del	1	Cost of one deletion
"cost_subst"	.cost_subst	1	Cost of one substitution
"max_cost"	.max_cost	5	Max. cost
"max_del"	.max_del	5	Max. number of deletions
"max_ins"	.max_ins	5	Max. number of insertions
"max_subst"	.max_subst	5	Max. number of substitutions
"max_err"	.max_err	5	Max. number of ins+del+subst

If the array *costs* is provided but contains none of the above indexes, the default values are used.

Return value

The **amatch()** function returns 1 on a successful match, 0 on a failure to match and -1 if *regex* is invalid (with TRE's error message in *ERRNO*) .

Obtaining match summary data

If a **third array argument** is provided to **amatch()**, and a match was successful, information about the match is returned via (clearing and) filling members of the *costs* array with these indexes:

Array index	Meaning
"cost"	Total cost of the match (Levenshtein distance)

```
"num_ins"     Total number of insertions
"num_del"     Total number of deletions
"num_subst"   Total number of substitutions
```

Obtaining parenthetical submatches

If an array (or empty Gawk variable symbol) is provided as the **fourth argument** , and a match is successful, the array will be cleared and filled with submatches corresponding to the parenthetical sub-expression in *regex*, with indexes *1...n*, up to a maximum of 20. The array member indexed by *0* will be the portion of *str* matched by the whole of *regex*.

A note on bytes and characters: While the **amatch**() function is roughly equivalent to the Gawk *match()* function, submatches are not returned as in *match()*, e.g. via *[i,“start”]* position and *[i,“length”]* (see Gawk `man` page). Instead only the literal substring for each parenthetical match is given. Gawk is multibyte aware, and *match()* works in terms of characters, not bytes, but TRE is byte-based, not character-based. Using the *wchar_t* versions of TRE functions cannot help if the input is a mix of single and multibyte characters. A simple Gawk routine must be used on the output array (*submatches*), if positions and lengths of the substrings are needed. E.g.:

```
print "i", "substring", "posn", "length"
p = 1
for (i = 1; i < length(submatches); i++) {
  idx = index(substr(str, p), submatches[i])
  len = length(out[i])
  print i, submatches[i], idx+p-1, len
  p = p + idx + len
}
```

EXAMPLE

```
@load "aregex"
BEGIN {
  str = "abcdfbc"
  regex = "^a(bc)d()(f)$"
  costs["max_cost"] = 6
  costs["cost_ins"] = 2
  if (amatch(str, regex, costs, submatches)>0)
    print costs["cost"], submatches[1]
}
```

SEE ALSO

The Gawk extension lib: https://sourceforge.net/projects/gawkextlib/ and TRE library: https://laurikari.net/tre/

AUTHORS

Cam Webb <cw@camwebb.info>, @laurikari for the TRE library, the *gawkextlib* authors

COPYING PERMISSIONS