NAME

sim − find similarities in C, Java, Pascal, Modula-2, Lisp, Miranda, or text files

SYNOPSIS

sim_c [−[**defFiMnpPRsST**] −**r** *N* −**t** *N* −**w** *N* −**o** *F*] file ... [/ [file ...]]
sim_c ...
sim_java ...
sim_pasc ...
sim_m2 ...
sim_lisp ...
sim_mira ...
sim_text ...

DESCRIPTION

Sim_c reads the C files *file ...* and looks for segments of text that are similar; two segments of program text are similar if they only differ in layout, comment, identifiers and the contents of numbers, strings and characters. If any runs of sufficient length are found, they are reported on standard output; the number of significant tokens in the run is given between square brackets.

Sim_java does the same for Java, *sim_pasc* for Pascal, *sim_m2* for Modula-2, *sim_mira* for Miranda, and *sim_lisp* for Lisp. *Sim_text* works on arbitrary text; it is occasionally useful on shell scripts.

The program can be used for finding copied pieces of code in purportedly unrelated programs (with −**s** or −**S**), or for finding accidentally duplicated code in larger projects (with −**f**).

If a / is present between the input files, the latter are divided into a group of "new" files (before the /) and a group of "old" files; if there is no /, all files are "new". Old files are never compared to each other.

Since the similarity tester reads the files several times, it cannot read from standard input.

There are the following options:

−d The output is in a diff(1)-like format instead of the default 2-column format.

−e Each file is compared to each file in isolation; this will find all similarities between all texts involved, regardless of duplicates.

−f Runs are restricted to segments with balancing parentheses, to isolate potential routine bodies (not in text).

−F The names of routines in calls are required to match exactly (not in text).

−i The names of the files to be compared are read from standard input, including a possible /; the file names must be one to a line. This option allows a very large number of file names to be specified; it differs from the @ facility provided by some compilers in that it handles file names only, and does not recognize option arguments.

−M Memory usage information is displayed on standard error output.

−n Similarities found are only summarized, not displayed.

−o F The output is written to the file named *F*.

−p The output is given in similarity percentages; see below; implies −**e** and −**s**.

−P As −**p** but more extensive; implies −**e** and −**s**.

−r N The minimum run length is set to *N* units; the default is 24 tokens, except in *sim_text*, where it is 8 words.

−R Directories in the input list are entered recursively, and all files they contain are involved in the comparison.

−s The contents of a file are not compared to itself (−s for "not self").

−S The contents of the new files are compared to the old files only − not between themselves.

−t N In combination with the **−p** option, sets the threshold (in percents) below which similarities will not be reported; the default is 1, except in *sim_text*, where it is 20.

−T A more terse and uniform form of output is produced, which may be more suitable for postprocessing.

−w N The page width used is set to *N* columns; the default is 80.

−− (A secret option, which prints the input as the similarity checker sees it, and then stops.)

The **−p** option results in lines of the form

```
        F consists for x % of G material
```

meaning that `x` % of `F`'s text can also be found in `G`. Note that this relation is not symmetric; it is in fact quite possible for one file to consist for 100 % of text from another file, while the other file consists for only 1 % of text of the first file, if their lengths differ enough. Each file is reported only once in the position of the `F` in the above line. This simplifies the identification of a set of files *A[1] ... A[n]*, where the concatenation of these files is also present. This restriction can be lifted by using the **−P** option instead. A threshold can be set using the **−t** option; this option is ignored under **−P**. Note that the granularity of the recognized text is still governed by the **−r** option or its default.

Sim_text accepts s p a c e d t e x t as normal text.

The program can handle UNICODE file names under Windows. This is relevant only under the **−R** option, since there is no way to give UNICODE file names from the command line.

Care has been taken to keep all internal processes linear in the length of the input, with the exception of the matching process which is almost linear, using a hash table; various other tables are used for speed-up. If, however, there is not enough memory for the tables, they are discarded in order of unimportance, under which conditions the algorithms revert to their quadratic nature.

EXAMPLES

The call

```
        sim_c *.c
```

highlights duplicate code in the directory. (It is useful to remove generated files first.) A call

```
        sim_c -f -F *.c
```

can pinpoint them further.

A call

```
        sim_text -e -p -s new/* / old/*
```

compares each file in `new/*` to each subsequent file in `new/*` and `old/*`, and if any pair has more that 20% in common, that fact is reported. Usually a similarity of 30% or more is significant; lower than 20% is probably coincidence; and in between is doubtful.

A call

```
        sim_text -e -n -s -r100 new/* / old/*
```

compares the same files, and reports large common segments. Both approaches are good for plagiarism detection.

LIMITATIONS

Repetitive input is the bane of similarity checking. If we have a file containing 4 copies of similar text,

A1 A2 A3 A4

where the numbers serve only to distinguish the similar copies, there are 7 similarities: A1=A2, A1=A3, A1=A4, A2=A3, A2=A4, A3=A4, and A1A2=A3A4, even discarding the overlapping A1A2A3=A2A3A4. Of these, only 3 are meaningful: A1=A2, A2=A3, and A3=A4. And for a table with 20 lines similar to each other, not unusual in a program, there are 715 similarities, of which at most 19 are meaningful. Reporting all 715 of them is clearly unacceptable.

To remedy this, finding the similarities is performed as follows: For each position in the text, the largest segment is found, of which a non-overlapping copy occurs in the text following it. That segment and its copy are reported and scanning resumes at the position just after the segment. For the above example this results in the similarities A1A2=A3A4 and A3=A4, which is quite satisfactory, and for N similar segments roughly *log N* messages are given.

A drawback of this heuristic is that the output is sensitive to the order of the input files. If we have two files
  file1 = A1, file2 = A2A3
then the order "file1 file2" gives "A1=A2, A2=A3" and "file2 file1" gives "A2=A3, A3=A1"; but both reports convey the same information.

BUGS

Since it uses *lex(1)* on some systems, it may crash on any weird construction that overflows *lex*'s internal buffers.

AUTHOR

Dick Grune, Vrije Universiteit, Amsterdam; dick@dickgrune.com.