```
#!/usr/bin/perl

# Goal: given an ontology, find the concepts in it which are opposites of each other.
# Definition of "opposites of each other:" see my paper draft.

# Usage:
# findAntonyms.pl infile_name

# ASSUMPTION: Input file is in .obo format.

# ASSUMPTION: word-medial contrasts are not handled yet.
# example: photosensitive/photoinsensitive

# TEST INPUT FILE: code/testFindAntonyms.obo
# TEST GOLD STANDARD OUTPUT FILE: code/testData/testFindAntonyms.tsv

# TEST CASES:
# - ambiguous prefix, is negative, opposite exists (cancerous, noncancerous) (anion,
ion, cation) (this is actually multiple opposites, I think)
# - ambiguous affix, is negative, no opposite exists
# - ambiguous, false opposite (none, one)
# - nonambiguous prefix
# - nonambiguous prefix, other word with shared substring but different prefix exists
(fan, ban)
# - nonambiguous prefix, other word with shared substring minus prefix exists (fan,
an)
# - lexicalized (malignant, benign)
# - multi-word: prefixal, adjectival, nominal, verbal...
# - opposite of term is a term
# - opposite of term is in a synonym
# - opposite of synonym is in a synonym
# - preceding, with different kinds of synonyms (exact, etc.)
# - word is in list of known negative words from Cohen et al. 2017
# - word is in list of known non-negative words from Cohen et al. 2017
# - word is in WordNet
# - word is not in WordNet


# GOAL: opposites that are formed not by addition of an affix,
# but by replacement of one affix with another.  Examples:
# hyperlipidemia <=> hypolipidemia
# hydrophobic <=> hydrophylic
sub replacements { # question: why doesn't this work with parentheses, while every
other function seems to??
  my $input = $_[0];
  my $output = $input;

  $DEBUG && print "Input to replacements(): <$input>\n";

# enlarged versus small

  my %paired_affixes = ("pro" => "anti",
                        "anti" => "pro",
                        "phylic" => "phobic",
                        "phobic" => "philic",
                        "hyper" => "hypo",
                        "hypo" => "hyper",
                        "pre" => "post",
                        "post" => "pre",
```

```
                        "emia" => "openia"); # will miss -penia with any other vowel
in front of it, but not sure that happens


  my @affixes = keys(%paired_affixes);
  foreach my $affix (@affixes) {
    my $replacement = $paired_affixes{$affix};
    # XXX TODO make this case-insensitive 'cause terms might start with an UC letter
    if ($output =~ s/^$affix/$replacement/ || $output =~ s/$affix$/$replacement/) {
      1 && print "HIT in replacements(): <$input> <$output>\n";
      # OK, we found an affix to replace. But, is the resulting term in the ontology?
      if ($gene_ontology_terms{$output}) {
        1 && print "$output is in the ontology!\n";
        storePair($input, $output);
      } # close if-term-is-in-ontology
    } # if you-did-a-replacement
  }
  return 1; # just to signal success
} # close function definition replacements()
############################################################
#################### SET THINGS UP #########################
############################################################

# catch typos (haha)
use strict 'vars';

# set to 1 for debugging output, to 0 to suppress same

my $DEBUG = 0;
#my $DEBUG = 1;

# set this to 1 if you only want the IDs--otherwise, to 0
my $IDS_ONLY = 0;

# set this to 1 if you're producing output for manual annotation. otherwise, set to 0
my $ANNOTATION_TRAINING = 0;

# used for drawing random samples from a list of distractors/quality control pairs
use List::Util 'shuffle';
srand(1789); # i hate not setting a seed; this is good for develpment, of course, but
to actually use it for multiple annotators, you need a different set of quality
control pairs every time--or just use the same one for everyone??

# TODO: change this name--we're now dealing with arbitrary ontologies, not just the
Gene Ontology
my %gene_ontology_terms = ();

# because opposition is bidirectional, you end up with Opposite(A, B) and Opposite(B,
A) as distinct
# pairs if you don't explicitly avoid it...
my %uniquePairs = ();

# store the IDs, which sometimes you want to output, and sometimes you don't.
# TODO: make a single hash that stores terms, IDs, and everything else--probably key
off of the terms; since they're
# what I work with the most, let's make it easy to get them with the keys() function
my %ontology_ids = ();


############################################################
```

```
################ READ IN YOUR ONTOLOGY #################
########################################################

my $infile = pop(@ARGV);
$DEBUG && print "Infile: $infile\n";

open (IN, $infile) || die "Couldn't open infile: $!\n";

# for parsing terms
my $id = undef;
my $term = undef;

while (my $line = <IN>) {
      # here we're assuming a single word only--TODO

        # note that regarding PR, it might not be worth looking. I tried grepping for
names that include "mutant" or "mutated", and only found 11...
        # Note: I was thinking about passing this in on the command line as a
parameter. However: because of the way that the OBO ontologies are constructed, you
can end up with IDs from multiple ontologies in the same .obo file. So: stick with
the long list...
        if ($line =~
/^id:\s+((BFO|RO|MP|SNOMED|PR|GO|CL|UBERON|NCBITaxon|PATO|IntAct:EBI-
|MI|MOD|NCBIGene|OGMS|OMIM|PR|ReTO|SIO|UniProt|HP):\d+)$/o) {
          $id = $1;
          $DEBUG && print "ID: <$id>\n";
        } # close if-ID

      # in some .obo files, I'm seeing "term", while it's "name" in others...
        if ($line =~ /^(name|term):\s+(.*)$/) { # Question: does greediness really
work the way that I'm wanting it to here, or can I end up with leading whitespace?
x+y* is a weird construction...

          $term = lc($2); # lc() is "lowercase"
          $DEBUG && print "term: $term\n";
        } # close if-name-or-term

        # once you have both a term and an ID, it's time to store away this concept.
        if ($id && $term)  {
          $gene_ontology_terms{$term} = 1;
          $ontology_ids{$term} = $id;
          $DEBUG && print "Just read in $id $term\n";
          undef($id); undef($term);
        }
} # END READING IN THE ONTOLOGY

########### define the things that indicate antonymy/oppositeness #############
my @negative_affixes = ("ab", "mis", "ar", "de", "non", "non-", "no", "anti", "un",
"in", "dys", "dis", "a", "an", "in", "im", "ir", "il", );


my %opposites = ("fast" => "slow",
                 "slow" => "fast",
                   "heavy" => "light",
                   "light" => "heavy",
                   "high" => "low",
                   "low" => "high",
                   "down" => "up",
                   "up" => "down",
                   "hot" => "cold",
```

```
"cold" => "hot",
"old" => "young",
"young" => "old",
"early" => "late",
"late" => "early",
"dark" => "light",
"light" => "dark",
"night" => "day",
"day" => "night",
"nighttime" => "daytime",
"daytime" => "nighttime", # with hyphens??
"opaque" => "transparent",
"transparent" => "opaque",
"higher" => "lower",
"lower" => "higher",
"raise" => "lower",
"lower" => "raise",
"improved" => "worsened",
"worsened" => "improved",
"complex" => "simple",
"simple" => "complex",
"partial" => "complete",
"complete" => "partial",
"forward" => "backward",
"backward" => "forward",
"forwards" => "backwards",
"backwards" => "forwards",
"dry" => "wet",
"wet" => "dry",
"hard" => "soft",
"soft" => "hard",
"large" => "small",
"small" => "large",
"big" => "little",
"little" => "big",
"positive" => "negative",
"negative" => "positive",
"positively" => "negatively",
"negatively" => "positively",
"odd" => "even",
"even" => "odd",
"early" => "late",
"late" => "early",
"dorsal" => "ventral",
"ventral" => "dorsal",
"somatic" => "germline",
"somatic" => "germ-line",
"germline" => "somatic",
"germ-line" => "somatic",
"increase" => "decrease",
"decrease" => "increase",
"increased" => "decreased",
"decreased" => "increased",
"stimulation" => "repression",
"repression" => "stimulation",
"stimulating" => "repressing",
"repressing" => "stimulating",
"stimulated" => "repressed",
"repressed" => "stimulated",
"mutant" => "wild type", # CAUTION: two-token phrase...
```

```
                "wild type" => "mutant",
                "mutant" => "wild-type",
                "wild-type" => "mutant",
                "acute" => "chronic",
                "chronic" => "acute",
                "back" => "front",
                "front" => "back",
                "open" => "closed",
                "closed" => "open",
                "fertile" => "sterile",
                "sterile" => "fertile",
                "elongated" => "shortened",
                "shortened" => "elongated",
                "rough" => "smooth",
                "smooth" => "rough",
                "absent" => "present",
                "present" => "absent",
                "primary" => "secondary",
                "secondary" => "primary",
                "secondary" => "tertiary", # from here on, it gets a bit fuzzy
                "tertiary" => "quaternary",
                "hyper" => "hypo"); # hyper and hypo are tough ones 'cause they're
prefixes but my code for these requires word boundaries BUT these aren't negatives,
per se.  Solution: a small function for this

###### The ontology has been read in--now look for the opposites ##########

# This is for training/crowdworking data only (or mostly ;-)):
# output some distractors/quality control pairs
if ($ANNOTATION_TRAINING) {
  outputQC();
}

#########################################################
################## FIND THE OPPOSITES ###################
#########################################################

foreach my $term (keys(%gene_ontology_terms)) {


  $DEBUG && print "Looking for antonyms of $term\n";

  ### AFFIXAL opposition--phosphorylate/dephosphorylate
  foreach my $affix (@negative_affixes) {

      my $candidate_antonym = $affix . $term;
        $DEBUG && print "Candidate: $candidate_antonym\n";


        # is the candidate in the ontology, too?
        if ($gene_ontology_terms{$candidate_antonym}) {
        $DEBUG && print "$candidate_antonym is an antonym of $term\n";
          storePair($term, $candidate_antonym);
     }
  }

  my $candidate_antonym = produceLexicalOpposite($term);
  # this evaluates to positive if the candidate antonym is already in the ontology...
  if ($gene_ontology_terms{$candidate_antonym}) {
    $DEBUG && print "Found one: $term, $candidate_antonym\n";
```

```
    storePair($term, $candidate_antonym);
  }

  replacements($term); # stuff like hyperlipidemia/hypolipidemia

  # XXXXXXX WHAT THE FUCK IS THIS DOING???
  if ((my $candidate_antonym = lexicalizedOpposites($term)) != 0) {
    #print "name:\t$term\n";
    #print "antonym:\t$candidate_antonym\n";
    #printTSV($term, $candidate_antonym);
    storePair($term, $candidate_antonym);
  }
}

##############################################
######### PRODUCE YOUR OUTPUT ##############
##############################################

$DEBUG && print "PRODUCING OUTPUT...\n";

# output the unique pairs only
my @unique_pairs = sort(keys(%uniquePairs));
foreach my $pair (@unique_pairs) {

  # these are *terms*
  my ($concept01, $concept02) = split("---", $pair);

  # if you just want to output the IDs, or just the terms, or...
  if ($IDS_ONLY) {
    print "$ontology_ids{$concept01}\t$ontology_ids{$concept02}\n";
  } elsif (1) {
    print
"$concept01\t$ontology_ids{$concept01}\t$concept02\t$ontology_ids{$concept02}\n";
  } else {
    printTSV(split("---", $pair));
  }

}

### FUNCTION DEFINITION: twoWordTerms()
### XXX TODO: WRITE A RECURSIVE FX INSTEAD, DUMMY
sub twoWordTerms() {
      my ($firstWord, $secondWord) = split($_);

      print "First word: <$firstWord> Second word: <$secondWord>\n";
        # TODO
} # close function definition twoWordTerms()

# FUNCTION DEFINITION: lexicalizedOpposites()
# assumption: no more than one opposite to find per term,
# since this returns as soon as one is found
sub lexicalizedOpposites() {

  my $candidate_term = $_;
  $DEBUG &&  print "Input to lexicalizedOpposites(): $candidate_term\n";

  if ($opposites{$candidate_term}) { return $opposites{$candidate_term}; }
  foreach my $opposite (keys(%opposites)) {
    $DEBUG && print "OPPOSITE BEING TESTED: $opposite\n";
    if ($candidate_term =~ /\b$opposite\b/) {
```

```
      $candidate_term =~ s/$opposite/$opposites{$opposite}/;
      return $candidate_term;
    }
  } # close foreach loop through opposites

  # you only reach this point if no potential opposites were found
  return 0;
} # close function definition lexicalizedOpposites()

### FUNCTION DEFINITION: printTSV() (PRODUCE TSV-FORMATTED OUTPUT)
# function to produce output, since I'm currently outputting from more than one place
in the code
sub printTSV() {
     my ($term, $antonym) = @_;

       # the leading tab is there to create a column for the manual annotation in a
spreadsheet
       print("\t" . $term . "\t" . $antonym . "\n");
}

### FUNCTION DEFINITION: outputQC() (ANNOTATOR QUALITY CONTROL/DISTRACTOR PAIRS)
# function to produce some distractors/QC pairs
sub outputQC() {

  # create a set of these, some of which definitely are opposites, and some of which
definitely aren't
  my %control_pairs = ("alive" => "dead", # OPP-LEX
                       "heavy traffic" => "light traffic", # OPP-LEX
                       "light traffic" => "heavy traffic", # OPP-LEX
                       "clean" => "dirty", # OPP-LEX
                       "dirty" => "clean", # OPP-LEX
                       "complex" => "simple", # OPP-LEX
                       "complex sugars" => "simple sugars", # OPP-LEX
                       "extant species" => "extinct species", # OPP-LEX
                       "melt" => "freeze", # OPP-LEX
                       "freeze" => "melt", # OPP-LEX

                       "acetylate" => "deacetylate", # OPP-MORPH
                       "illiterate" => "literate", # OPP-MORPH
                       "asymptomatic" => "symptomatic", # OPP-MORPH
                       "unsympathetic" => "sympathetic", # OPP-MORPH
                       "locking" => "unlocking", # OPP-MORPH
                       "cation" => "anion", # OPP-MORPH
                       "clean" => "unclean", # OPP-MORPH
                       "underweight" => "overweight", # OPP-MORPH
                       "dephosphorylation" => "phosphorylation", # OPP-MORPH
                       "ectomorph" => "endomorph", # OPP-MORPH
                       "vertebrate" => "invertebrate", # OPP-MORPH

                       "bubble" => "bubble", # IDENT
                       "cat" => "cat", # IDENT
                       "unclean" => "unclean", # IDENT
                       "asymptomatic" => "asymptomatic", # IDENT

                       "dog" => "dogged", # RELATED PAIR
                       "dog" => "dogs", # RELATED PAIR
                       "Dog" => "God", # RELATED PAIR
                       "God" => "Dog", # RELATED PAIR

                       "ontology" => "manuscript", # UNRELATED PAIR
```

```
                        "pear" => "wolf", # UNRELATED PAIR
                        "aorta" => "orca", # UNRELATED PAIR
                        "onion" => "union", # UNRELATED PAIR

                        "dead" => "ad", # DELETION TO REAL WORD
                        "amen" => "men", # DELETION TO REAL WORD
                        "union" => "ion", # DELETION TO REAL WORD

                        "apple" => "pple"); # DELETION TO NON-WORD
                        "ill" => "l", # DELETION TO NON-WORD
                        "cle" => "uncle", # DELETION TO NON-WORD
  # get a random sample of those pairs
  my @keys = keys(%control_pairs);

  # pull n pairs from there
  my $sample_size = 5; # only need this if you want to give multiple annotators each
a different set of QC pairs
  # see above for the library that I got shuffle() from
  my @shuffled = shuffle(@keys);
  my @sorted = sort(@keys); # contradicts the previous line, obviously!
  # for when you want less than the whole set...
  #my @sample = @shuffled[0, $sample_size];
  #my @sample = @shuffled; # gives you the whole set
  my @sample = @sorted; # the opposite of the shuffling, obviously!
  $DEBUG && print "random sample:\n\n";
  foreach my $key (@sample) {
    print "\t$key\t$control_pairs{$key}\n";
  }
  $DEBUG && print "end of random sample...\n";
} # close function definition outputQC()

### FUNCTION DEFINITION: storePair()
# because antonymy is bidirectional, you end up with duplicates, so I will put them
in a hash to avoid that
# TODO: sort the pair so that the earliest one in alphabetical order is first in the
pair--easier to deal with
# in the long run...
sub storePair() {
  my @pair = @_;
  @pair = sort(@pair); #
  my $this_pair = $pair[0] . "---" . $pair[1];
  $uniquePairs{$this_pair}++;
}

### FUNCTION DEFINITION: produce a lexical opposite, if there be one
sub produceLexicalOpposite() {
  my $input = $_[0]; # ugly syntax for "the first argument passed to the function"

  # there's probably a much, much faster way to do this--certainly in R...
  foreach my $lexical_opposite (keys(%opposites)) {
    if ($input =~ s/\b$lexical_opposite\b/$opposites{$lexical_opposite}/) {
      return($input); # ...which is now hopefully the opposite of what it used to be!
    }
  }

  # if you got here, you didn't find anything...
  return 0;
} # close function definition
```