```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ABOUT THIS DOCUMENT

; Here are some notes on using Timm’s lisp code.

; In the following definition of a table of data "deftable"
; starts a new table and "!" writes a new row of data into
; that table.

(deftable  weather forecast temp humidty windy !play)

; in deftable, !xx denotes a class (the dependent variable).
; everything else are the independent variables and are of two
; types: "sym" (for symbol) and "num" (for numeric).
; $xx denotes a "num" attribute. All other attributes are "sym".

(! sunny    hot  high   FALSE no)
(! sunny    hot  high   TRUE  no)
(! overcast hot  high   FALSE yes)
(! rainy    mild high   FALSE yes)
(! rainy    cool normal FALSE yes)
(! rainy    cool normal TRUE  no)
(! overcast cool normal TRUE  yes)
(! sunny    mild high   FALSE no)
(! sunny    cool normal FALSE yes)
(! rainy    mild normal FALSE yes)
(! sunny    mild normal TRUE  yes)
(! overcast mild high   TRUE  yes)
(! overcast hot  normal FALSE yes)
(! rainy    mild high   TRUE   no)

; What data structures are needed to store the above?
; There are two answers to this question.
; For proj1a 1b 1c, the answer is "it does not matter".

; But for project2, you need to understand this stuff since
; the following code is a backbone system (on top of which,
; you can build which2).

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; GETTING THIS CODE

; The following code is an assembly of stuff that you can find at
;
; cd $HOME/svns/csx73/lisp101
; svn export http://unbox.org/wisp/var/timm/10/dm/lisp101/ml

; To load this code
; cd $HOME/svns/csx73/lisp101/ml
; emacs boot.lisp
; ; then load boot.lisp into SLIME

; To modify this code
; ; edit boot.lisp
; ; add in your own files

; FILE LIST
; ---------
; abcd.lisp    ; util, ignore, for now
; bestof.lisp  ; util
; boot.lisp    ; list of files
; data.lisp    ; routines for deftable and "!"
; structs.lisp ; defines structs and the *W* variable
; which2.lisp  ; sample code to get you started with proj2


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; HIGH-LEVEL STUFF

;When the above is loaded, there is a global *w* storing the result.
;THis global is of type "wme".

;; from structs.lisp
```

```
(defparameter *w* nil)
(defun w0 () (setf *w* (make-wme)))

;; from data.lisp
(defun data (&optional f)
  (w0)                        ; reset the global *w*
  (load (or f (thefile)))     ; load the file, or the default file
  (funcall (wme-ready *w*))   ; prep
  (funcall (wme-run *w*))     ; learn
  (funcall (wme-report *w*))  ; report
)

; For example

> (data "../data/discrete-lisp/weather.lisp")
T
> (thetable)

#S(TABLE
   :NAME WEATHER
   :ROWS (#S(ROW
             :CELLS (SUNNY HOT HIGH TRUE NO)
             :CLASS NO
             :UTILITY 0
             :SORTKEY 0.05486242722621558d0)
          #S(ROW
             :CELLS (RAINY MILD HIGH TRUE NO)
             :CLASS NO
             :UTILITY 0
             :SORTKEY 0.15162094871921356d0)
          #S(ROW
             :CELLS (SUNNY MILD HIGH FALSE NO)
             :CLASS NO
             :UTILITY 0
             :SORTKEY 0.16896725912164381d0)
          #S(ROW
             :CELLS (SUNNY HOT HIGH FALSE NO)
             :CLASS NO
             :UTILITY 0
             :SORTKEY 0.3374376731603196d0)
          #S(ROW
             :CELLS (RAINY COOL NORMAL TRUE NO)
             :CLASS NO
             :UTILITY 0
             :SORTKEY 0.34725111281542576d0)
          #S(ROW
             :CELLS (OVERCAST HOT NORMAL FALSE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.0762189353172944d0)
          #S(ROW
             :CELLS (SUNNY MILD NORMAL TRUE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.1621534186087104d0)
          #S(ROW
             :CELLS (RAINY MILD NORMAL FALSE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.2642953673986543d0)
          #S(ROW
             :CELLS (SUNNY COOL NORMAL FALSE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.2826687920405857d0)
          #S(ROW
             :CELLS (OVERCAST COOL NORMAL TRUE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.3493395062490854d0)
          #S(ROW
             :CELLS (OVERCAST HOT HIGH FALSE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.3827964523910197d0)
          #S(ROW
             :CELLS (RAINY COOL NORMAL FALSE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.4209536138992274d0)
          #S(ROW
             :CELLS (OVERCAST MILD HIGH TRUE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.4224535227656765d0)
          #S(ROW
             :CELLS (RAINY MILD HIGH FALSE YES)
             :CLASS YES
             :UTILITY 0
             :SORTKEY 1.4498501279678888d0))
   :KLASSES (#S(KLASS :NAME NO :N 5) #S(KLASS :NAME YES :N 9))
   :COLS (#S(SYM :NAME FORECAST :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME TEMP :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME HUMIDTY :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME WINDY :GOALP NIL :COUNTS {hash of 0 items})
          #S(SYM :NAME !PLAY :GOALP #\! :COUNTS {hash of 0 items}))
   :RESULTS NIL)

; My code has a bunch of accessors to simplify getting to "the" last
; table loaded:

(defmacro thetable   ()        ‘(wme-table *w*))
(defmacro thecols    (&optional tbl) ‘(table-cols    (or ,tbl (wme-table *w*))))
(defmacro thename    (&optional tbl) ‘(table-name    (or ,tbl (wme-table *w*))))
(defmacro therows    (&optional tbl) ‘(table-rows    (or ,tbl (wme-table *w*))))
(defmacro theklasses (&optional tbl) ‘(table-klasses (or ,tbl (wme-table *w*))))

; But you can’t understand the code unless you look under the hood
; at the structs they came from. So....

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Under the hood

;; from structs.lisp
(defstruct wme
  (goal             #\!)
  (num              #\$)
  (unknown          #\?)
  (file             "../discrete-lisp/weather.lisp")
  (utility-function #’zero)
  (!                #’defrow)
  (ready            #’sort-rows)
  (run              #’noop)
  (report           #’noop)
  table
)

#| you won’t get the above unless you know the data structures


STRUCTURES
==========

Wme with
    goal    = char ; if col.name has this char, then this is a class
    num     = char ; if col.name has this char, then this is a numeric column
    unknown = char; if any item in row.cells is this char then this value is unk
nown
    file    = string; place to load a file
    !       = thing to be called when we see "(! a d c )"
    ready   = thing to do to prep the table
    run     = thing to do to process the table
    report  = thing to do to report the result
    table   = Table

Table with
```

```
      name    = atom
      rows    = list of  Row
      klasses = list of Klass ;only one per class in rows
      cols    = list of Col
      results = list of Result

Row with
    cells   = list of atom ; and #cells = #cols
    class   = atom
    utility = number
    sortKey = number

Klass with
      name  = atom ;
      n     = number ; stores how many rows with this Klass name in Table

Col with
    name isa atom
    goalp isa boolean; true if this is a class attribute


Sym isa Col with
    counts = hashtable ; counts of (value in column in class)

Num isa Col with
    n = number
    sum = number
    sumsq = number
    min = number
    max = number

|#

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; How to count  the frequencies in the above table?
; Note- if you understand the above structures, this code
;       will make sense to you

;; from which2.lisp

(defun train (tbl)
   (dolist (row (therows tbl))
     (how-manys (thecols tbl)    ; get the column headers
                (row-cells row)  ; get the cells
                (row-class row)  ; get the class of this row
                )))

(defun how-manys (cols cells class)
  (labels ((worker (col cell)
             (how-many class
                       (col-name col)
                       cell
                       (sym-counts col))))
    (mapcar #’worker cols cells))) ; run down cols and cells in parallel

(defun how-many (class what cell hash)
  (when (knownp cell) ; skip any cell labelled "?"
    (inch ‘(,class   ,what ,cell) hash)
    (inch ‘(,*every* ,what ,cell) hash)))

(defun inch (key hash)
  "increment a hash bucket from zero"
  (incf (gethash key hash 0)))

(defun !how-manys1 ()
  (reset-seed)
  (data "../data/discrete-lisp/weather.lisp")
  (train (thetable))
  (with-output-to-string (s)
    (dolist (col (thecols))
      (showh (sym-counts col) :stream s))))

> (!how-manys1)
```

```
       "(ALLQZJX FORECAST OVERCAST) = 4
        (ALLQZJX FORECAST RAINY) = 5
        (ALLQZJX FORECAST SUNNY) = 5
        (NO FORECAST RAINY) = 2
        (NO FORECAST SUNNY) = 3
        (YES FORECAST OVERCAST) = 4
        (YES FORECAST RAINY) = 3
        (YES FORECAST SUNNY) = 2
        (ALLQZJX TEMP COOL) = 4
        (ALLQZJX TEMP HOT) = 4
        (ALLQZJX TEMP MILD) = 6
        (NO TEMP COOL) = 1
        (NO TEMP HOT) = 2
        (NO TEMP MILD) = 2
        (YES TEMP COOL) = 3
        (YES TEMP HOT) = 2
        (YES TEMP MILD) = 4
        (ALLQZJX HUMIDTY HIGH) = 7
        (ALLQZJX HUMIDTY NORMAL) = 7
        (NO HUMIDTY HIGH) = 4
        (NO HUMIDTY NORMAL) = 1
        (YES HUMIDTY HIGH) = 3
        (YES HUMIDTY NORMAL) = 6
        (ALLQZJX WINDY FALSE) = 8
        (ALLQZJX WINDY TRUE) = 6
        (NO WINDY FALSE) = 2
        (NO WINDY TRUE) = 3
        (YES WINDY FALSE) = 6
        (YES WINDY TRUE) = 3
        (ALLQZJX !PLAY NO) = 5
        (ALLQZJX !PLAY YES) = 9
        (NO !PLAY NO) = 5
        (YES !PLAY YES) = 9"


; What’s this "ALLQZKK" nonsense? Well, sometimes it is useful
; to count column range frequencies in EVERY class. So we make
; up a class name (something using the rarest letters- as defined
; by the point scores in SCRABBLE QZJX) and count all (column range)
; pairs in that majic EVERY class.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Using the above, lets sort all the ranges according
; to their ability to distinquish one class from all the others.

; In the following code, if there are N classes in the system,
; then we make each one the "target". Our goal then is to
; divide the data into
; a) "target1, rest1" (where "rest1" is everything  but "target1")
; b) "target2, rest2" (where "rest2" is everything  but "target2")
; etc

(defun learn (tbl report)
  (dolist (target (theklasses tbl))
    (learn1  target tbl report)
    ))

(defun learn1 (target tbl report)
  (let ((which (round0 target tbl)))
    (rounds target which tbl report)))

(defun round0 (target tbl)
  "returns a sorted list of triples (score col value)
   where SCORE is higher if (col value) is more common
   in the BEST target class rather than the REST"
  (let (out
        (n (length (therows tbl)))) ; the total number of rows is "n"
    (labels
        ((worker (hash want m ; the number of rows for this class is "m"
                       what class value &aux s)
           (if (eql class want)
             (if (setf s (b^2/b+r hash want m n what value))
                 (push (list (round s 0.01)
```

```
                             what value)
                       out)))))
      (dolist (col (thecols tbl))         ; for every column
        (unless (col-goalp col)           ; that’s not the goal
          (dokeys (key  (sym-counts col)) ; for everything counted in that col
            (worker (sym-counts col)      ; get the hash table counds
                    (klass-name target)   ; what class are we targetting?
                    (klass-n target)      ; how many of them do we have?
                    (col-name col)        ; what is the col name?
                    (first key)           ; what class is being counted?
                    (third key)           ; what value we looking at?
                    ))))
      (sort out #’> :key #’first))))

(defun b^2/b+r  (hash want m n what value)
  (let* ((every (gethash ‘(,*every* ,what ,value) hash 0))
         (b0    (gethash ‘(,want    ,what, value) hash 0))
         (r0    (- every b0))
         (b     (/ b0 m))        ; ratio in target
         (r     (/ r0 (- n m)))) ; ration everwhere else
    (if (> b r)                  ; in more better than rester
        (/ (* b b)               ; b^2/(b+r)
           (+ b r (randf 0.0000001)))))) ; add a pinch to dodge div/0 errors

;;; so does the above all work? well, we need a test rig

(defun which2 (&optional (tbl (thetable)) (report t))
  (train tbl)
  (learn tbl report)
  )

(defun rounds (class which tbl report)
  (declare (ignore tbl))
  (format report ";;; ~a~%" (klass-name class))
  (dolist (one which)
    (format report "     ~a~%" one)))

(defun !learn1 ()
  (reset-seed)
  (data "../data/discrete-lisp/weather.lisp")
  (with-output-to-string (s)
    (which2 (thetable) s)))

(deftest !learn ()
  (test (!learn1)
        ";;; NO
          (56 HUMIDTY HIGH) ;; best predictor for not playing golf
          (44 FORECAST SUNNY)
          (39 WINDY TRUE)
          (26 TEMP HOT)
          (22 FORECAST RAINY)
        ;;; YES
          (51 HUMIDTY NORMAL) ;; best predictor for playing golf
          (44 FORECAST OVERCAST)
          (42 WINDY FALSE)
          (23 TEMP MILD)
          (21 TEMP COOL)"))
```

```
     #!/sw/bin/gawk -f

     ##########################################################################
     #    This program is free software: you can redistribute it and/or modify
  5  #    it under the terms of the GNU Lesser General Public License as published by
     #    the Free Software Foundation, either version 3 of the License, or
     #    (at your option) any later version.
     #
     #    This program is distributed in the hope that it will be useful,
 10  #    but WITHOUT ANY WARRANTY; without even the implied warranty of
     #    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
     #    GNU Lesser General Public License for more details.
     #
     #    You should have received a copy of the GNU Lesser General Public License
 15  #    along with this program.  If not, see <http://www.gnu.org/licenses/>.
     ##########################################################################

     # which2d : a stochastic anytime rule learner  for discrete classes
     # (c) Tim Menzies (tim@menzies.us) 2010, LGLP 3.0
 20
     # This program builds rules by ranking ideas, then repeatedly building new ideas
     # by picking # and combining two old ideas (favoring those with higher ranks).
     New
     # ideas generated in this way are ranked and thrown back into the same pot as
     # the old ideas so, if they are any good, they might be picked and extended
 25  # in subsequent rounds. Alternatively, if the new idea stinks, it gets buried
     # by the better ideas and is ignore.

     # One important aspect of the following is that the scoring routines for
     # ideas are completely seperate from the rest of the code (see the "score1"
 30  # function). Hence, it is a simple # matter to try our different search biases.

     # e.g. This call produces the following output.
     # gawk -f which2.awk titanic.arff

 35  # In the following, the "candidates" are ideas that look promsing
     # and "score" ranks the candidates. If the max score does not improve
     # from the last round, then "lives" decreases.

     # Each round tries random combinations of the stuff from prior rounds
 40  # (favoring those things with higher scores). Hence, at round 1,
     # all the candidates are singletons. But. later on (see line 54)
     # the candidates can grow to combinations of things.

     # Each round prunes the candiates so that only the better candiadtes
 45  # surive to round+1.

     BEGIN  {
             Seed=1          # Random number see.
         More = 1.02;    # Improvement means at least a 2% growth
 50      Lives=5;        # If no improvement after five rounds, give up
         Dull=0.1;       # Ignore candidates with score < Dull*MaxScore
         Beam=10;        # Only the top (say) 10 candidates survive to the next round
         Samples=20;     # Pick this number of pairs of candidates from the last roun
     d
         Pinch = 1/1000; # Add a random number of up to "Pinch" to each score
 55          OverFitted=3;   # When do we prune a rule that matches on too few instan
     ces?
         CONVFMT="%.8g"; # Increase the string size for array contents so we can see
     the Pinch
         IGNORECASE=1;
         SUBSEP = "=";
         _ = SUBSEP
 60          OFS=",";
         C=","
         Verbose=1       # Verbose = 0 means silence
     }

 65  ## --------------------------------------------------
     #Data entry. Pretty routine stuff.
     /@attribute/ {Name[++Name[0]]=$2; Name[$2] = Name[0]}
                  {gsub(/[ \t]*/,"")} # no blanks
                  {gsub(/%.*/,"")}    # no comments
```

```
 70  /^$/         {next}               # no blank likes
     /@data/      {In=1;FS=","; srand(Seed)}
     /@/          {next}
     In           {Rows++;
                   train(All,H,Rows,Data,F,$NF)}
 75  END          { learn(All,H,Rows,Data,F) }

     function train(all,h,row,d,f,class,   what,i) {
         h[class]++
         for(i=1;i<=NF;i++)  {
 80          if ($i == "?")
                 continue;
             what = Name[i]
             d[row,what]=$i
             all[what,$i]++
 85          if (i != NF)
                 f[class,what,$i]++ }
     }

     # Now we can begin. Try learning rules for each hypothesis.
 90  function learn(all,h,rows,data,f,   class) {
         for(class in h)
             learn1(class,all,h,rows,data,f)
     }
     # In round0, offer a rough ranking of
 95  # the ranges. In subsequent rounds, randomly select and combine ranges
     # from prior randoms.
     function learn1(class,all,h,rows,data,f,   which0,which) {
         round0(class,all,rows,f,h,which0); # make some initial guess
         o(which0,"which0","-n -k 5")
100      rounds(class,1,0,Lives,which0,rows,data,f,which)
             #exit
     }

     # In round one, score by b^2/(b+r); i.e. things more likely in
105  # the target class than otherwise (with some support weighting)
     function round0(class,all,rows,f,h,which,  some,i,j,b,r,s,memo,score) {
         for(i in all) {
             some = f[class _ i]
             r    = (all[i] - some)/(rows  - h[class])
110          b    = some / h[class]
             if (b > r) {
                 j = class "," i
                 s = b^2/(b+r) + rand()*Pinch
                 memo[s] = j
115              score[j]= s
             }}
        chop(score,memo,which) # prune the dull candidates
     }

120  # Given some score[key]=number and memo[number]=key,
     # sort the scores and return the top Beam
     # number of keys, pruning all keys less than
     # Dull times the max score.
     function chop(score0,memo,out,   score,n,i) {
125      n=asort(score0,score)
         for(i=n;i>=1;i--) {
             if (score[i] <= score[n]*Dull)
                 break;
             if (i <= n - Beam)
130              break
             out[memo[score[i]]] = score[i]
         }
     }

135  # In subsequent rounds one, score all the candidates
     # by running that combination over the data (see the "score"
     # function. Note the "score" paramter that caches prior
     # calcuations of the score. This speeds up the code by
     # a factor of four (for large data sets).
140  function rounds(class,round, max0,lives,which0,rows,data,f,out,score,   \
                     max,i,sample,which1,s,memo,which2) {
         if (round == 1)
        max=0
    else { # terminate if we have stopped improving
        max = most(which0)
        lives =  (max > (max0*More)) ? Lives : lives - 1
        if(lives < 0) { # if termination, copy input to out
            for(i in which0)
                out[i] = which0[i]
            return max }
    }
        print "\n————————————————————————————————————————————————————"
    print "% class: " class " seed: " Seed \
          " round: " round " max: " max " lives: " lives
    normalize(which0)           # make all the counts n= 1..100
    explode(which0,sample)      # copy items n times
    twos(class,sample,Samples,which1) # pick items at random from that sample
    for(i in which0)            # add in the last rounds' ideas
        which1[i] = i;
    if (Verbose) values(which1,"candidate")
    for(i in which1) {          # score the new picks and the last rounds's idea
s
        s = (i in score) ? score[i] : score(class,i,rows,data,f) + rand()*Pinch
        memo[s] = i
        score[i] = s
        }
    chop(score,memo,which2)     # prune the dull candidates
    if (Verbose) o(which2,"score","–n –k 1")
    return rounds(class,round+1,max,lives,which2,rows,data,f,out,score)
}

## -----------------------------------------
# Randomly pick pairs and combine them. Note that,
# in the following code, the picks come from "sample"
# where an item may be repeated many times (so things
# that are often repeated are more likely to be picked).

# "n" times, pick two things from "sample"
# and store them in "sampled". note htat
# the combined rules all start with the target class
function twos(class,sample,n,sampled,  pair) {
    while(n--) {
        pair= two(class,sample)
        sampled[pair]=pair
        }
}

# Pick two things at random. Try not
# to pick the same thing twice. Return
# the combination of the two things you pick.
function two(class,sample, tries, this, that) {
    this = one(sample)
    if(tries == 9) # nine lives
        return this
    that = one(sample)
    if (this == that)
        return two(class,sample,tries + 1)
     else
        return combine(class,this,that)
}

# combine two rules. don't repeat any ranges.
# sort them so that all the ranges of the same
# feature fall together. Note that the frst item in
# a rule is the target class. prune those entries
# away (so they don not repeat themselves).
function combine(class,this,that,    n,i,used,tmp,out) {
    sub(/^[^,]*,/,"",this)
    sub(/^[^,]*,/,"",that)
    split(this "," that,tmp,",")
    n=asort(tmp)
    out=tmp[1]
    used[tmp[1]]=1
    for(i=1;i<=n;i++)
        if (!used[tmp[i]]) {
```

```
            out = out "," tmp[i]
            used[tmp[i]] = 1
        }
    return class "," out
}

## --------------------------------
## score a rule by finding its matching rows in data.
function score(class,rule,rows,data,f,
                         goal,row, col,a,b,c,d,triggered,pd,pf,prec,acc,support,
s,fits) {
    a=b=c=d=Pinch # stop divide by zero errors
    goal=Name[Name[0]]
    for(row=1;row<=rows;row++) {
        triggered = matched(row,data,rule)
        if (data[row,goal] == class) {
            if (triggered) {d++} else {b++}
        } else {
            if (triggered) {c++} else {a++}
        }
    }
    fits    = c + d
    pd      = d/(b+d)
    pf      = a/(a+c)
    prec    = d/(c+d)
    acc     = (a+d)/(a+b+c+d)
    support = (c+d)/(a+b+c+d)
    return score1(pd,pf,prec,acc,support,fits)
}
function score1(pd,pf,prec,acc,support,fits) {
    if (fits <= OverFitted)
       return 0
    if (Eval==1) return acc
    if (Eval==2) return 2 * pd * prec/(pd+prec)
    if (Eval==3) return 2 * pd * pf/(pd+pf)
    if (Eval==4) return support * 2 * pd * pf/(pd+pf)
    return support * 2 * pd * prec/(pd+prec)
}

# Given "this" of the form "f1_v1,f2_v2,...." see if "row" matches "this".
# Assumes that disjunctions are modeled as  configuous values from the
# same feature (this is gaurenteed by "combine"). Hence, whenever
# we move to a new feature, we need to check that at least of the values
# mentioned with the old feature was found.
function matched(row,data,this,    col,n,goals,pair,f0,f,status) {
    n=split(this,goals,",")
    for(col=2;col<=n;col++) {
        split(goals[col],pair,_)
        f = pair[1]
        status[f] += data[row,f] == pair[2]
        if (f0 && (f != f0) && !status[f0])
            return 0
        f0 = f
    }
    return status[f]
}

## -------------------------------------
## interesting utils

# Given an array a[i]=n,
# fill up "out" with "n" number
# of "i". This creates a "sample" of
# things, from which we can pick randomly
# biased by the relative frequencies of "n".
# The total size of the sample is stored
# in "sample[0]"
function explode(a, out,i,j) {
    for(i in a)
        for(j=1;j<=a[i];j++)
            out[++out[0]] = i
}
```

```
# Pick any item at random from "sample".
# Assumes that the same size is in array
# element "sample[0]"
function one(sample, any) {
    any = int(rand()* sample[0])+1
    return sample[any]
}

## -------------------------------------
## boring utils

# Given an array a[i]=num, normalize
# all the numbers to integers 0..100
function normalize(a, i,sum) {
    for(i in a) sum += a[i]
    for(i in a) a[i] = int(100*a[i]/sum)
}

# combine an feature/ range
function fv(f,v) { return f _ v }

# find the max item in an array
function most(a,   i,max) {
    max = -1000000000
    for(i in a)
        if (a[i] > max)
            max = a[i];
    return max
}

# print array values
function values(a,s,what,  i,com) {
        print ""
    com = what ? "sort " what : "sort"
    for(i in a)
        print "% " s": " a[i] | com;
    close(com)
 }
# print an array, sorted by "what"
function o(a,s,what,   i,com) {
        print ""
    com = what ? "sort –t, " what : "sort –t, "
    for(i in a)
        print "["a[i] ","i"]." | com;
    close(com)
 }
```