```
#! /usr/bin/python
# ANNOTATED VERSION
# WordCracker1.31a (MiniEntry) by Mike Sweeney (mjs) 28-feb-05
# Requires python version 2.2 or above (I think)
# 161 lines - 78 comment/space/dbug lines = 83 working lines
# Less that 3000 bytes without comments.

BC = '_'   # Use '-' for me, '_' for official entry

import sys, time, operator
from string import *
from random import randrange

# General purpose sequence summing function:
def sum(s): return reduce( operator.__add__, s, 0 )

# If "prog -w CAT DOG FRED" then use words from arg list:
if sys.argv[1] == '-w': words = [ upper(w) for w in sys.argv[2:] ]
# Otherwise read words from file:
else: words = [strip(a) for a in file( sys.argv[1] ).readlines() if strip(a)]

tlimit = time.clock() + 56.0

################### INDEXING ###################

# Set up character mapping dictionary for fast fragment lookup:
# cm[fragment]->[(wordlength,word,fragpos),...]
cm = {}
for w in words:
    for i in range(len(w)):
        # Index each character in word:
        cm.setdefault(w[i],[]).append((len(w),w,i))
        for j in range(2,8):
            if j<=i:
                # Index \w\s+\w fragments in word:
                pat = w[i-j]+(BC*(j-1))+w[i]
                cm.setdefault(pat,[]).append((len(w),w,i-j))

# Make mapping of word popularity: wpop[word]->int
wpop = dict([(w,sum([len(cm[ch]) for ch in w])) for w in words])

# For every entry in the index, add word popularity at the front of
# each record:
for k in cm: cm[k] = [(wpop[w],wl,w,i) for wl,w,i in cm[k]]

# Sort each record list in index. Most popular word first:
[ (a.sort(),a.reverse()) for a in cm.values() ]

# Make list of most popular character fragments:
# popchs->[(int,fragment),...]
popchs = [ (len(wset),ch) for ch,wset in cm.items() ]

# Sort lowest to highest popularity:
popchs.sort()

# Ranges for character in words, puzzle width, puzzle height:
rc, rw, rh = range(1,16), range(32), range(16)

################### findw FUNCTION ###################

def findw( q, px, py, pdx, pdy, slots ):
    '''Return zero, one or two "slots" that can be used for word placement
    at this position (px,py) and orientation (pdx). Check for preceeding
    space, space after, and a possible letter followed by more space.
    Returns [], or [ [x,y,dx,int,\w-+\w,int], [x,y,dx,int,\w,int] ], or
    just [ [x,y,dx,int,\w,int] ]. "q" is the query (get) method for the
    data structure of the puzzle.'''
    pat = q((px,py))
    # Find space before letter:
    dx,dy = -pdx,-pdy
```

```
    for spre in rc:
        x,y = px+dx*spre,py+dy*spre
        if q((x,y))!=BC or 'A'<=q((x+dx,y+dy))<='Z': break
        if 'A'<=q((x+dy,y+dx))<='Z' or 'A'<=q((x-dy,y-dx))<='Z': break
    # Find space after letter:
    dx,dy = pdx,pdy
    for spost in rc:
        x,y = px+dx*spost,py+dy*spost
        if q((x,y))!=BC: break
        if 'A'<=q((x+dy,y+dx))<='Z' or 'A'<=q((x-dy,y-dx))<='Z': break
        if 'A'<=q((x+dx,y+dy))<='Z':
            # If fwd space ends in a letter, add pat and find more:
            slots.append( [px,py,pdx, spre-1, pat, spost-1] )
            if 'A'<=q((x+dx*2,y+dy*2))<='Z' : pat=''
            else: pat = q((px,py))+(BC*spost)+q((x+dx,y+dy))
            break
    if len(pat)>1:
        for spost in rc:
            x,y = px+dx*(spost+len(pat)-1), py+dy*(spost+len(pat)-1)
            if q((x,y))!=BC or 'A'<=q((x+dx,y+dy))<='Z': break
            if 'A'<=q((x+dy,y+dx))<='Z' or 'A'<=q((x-dy,y-dx))<='Z': break
    if pat: slots = [[px,py,pdx, spre-1, pat, spost-1]] + slots
    return slots

########################## run FUNCTION ##########################

def run(sc=0):
    ''' Create a blank puzzle, and as long as there are pivots (places in
    the puzzle where another word can join), find a word to connect at
    a pivot, write the word in, and add to the score. Return the completed
    puzzle and the score. '''
    pz = dict( [((x,y),BC) for x in rw for y in rh] )
    # Place the most popular letter in the middle:
    pz[(15,8)] = popchs[-1][1]
    piv = { (15,8,1) : 1 }
    while piv:
        targ = None
        # Pick up to 16 pivots into pl and compute slots for them:
        pl = piv.keys()
        z = randrange( max( len(pl)-16, 1 ) )
        slots = [findw(pz.get,x,y,dx,1-dx,[]) for x,y,dx in pl[z:z+16]]
        # Order slots with complex patterns first:
        ps = [ s[0] for s in slots if len(s)==2 ]
        ps += [ s[-1] for s in slots if len(s)>=1 ]
        # If there are no prefered slots, the puzzle is done:
        if not ps: break
        for x,y,dx,spre,pat,spost in ps :
            dy = 1-dx
            if len(pat)>1 or spre+spost > 0 :
                # For each possible word placement that fits the pattern:
                for wpop,wlen,word,wpre in cm.get(pat,[]):
                    # If this word would fit in the space of the slot:
                    if wpre <= spre and wlen-wpre-len(pat) <= spost:
                        # We have a target word, so quit out
                        targ = word
                        break
                if targ: break
                # Delete this pivot (no word will fit it):
                if len(pat)==1 and piv.get((x,y,dx)) : del piv[(x,y,dx)]
        if targ:
            # Write the word into the puzzle:
            for i in range(len(targ)):
                cx,cy = x+dx*(i-wpre),y+dy*(i-wpre)
                if pz[(cx,cy)] != targ[i] :
                    # Write the letter in:
                    pz[(cx,cy)] = targ[i]
                    # Add this position to the pivot set:
                    piv[(cx,cy,dy)] = 1
                else:
                    # This letter was already here, so delete that pivot:
```

```
                    if piv.get((cx,cy,dx)) : del piv[(cx,cy,dx)]
            # Update score:
            sc += len(targ)
    return pz, sc


###################### MAIN() ######################

# Create puzzle solutions until time runs out:
top = -1
while 1:
    pz, sc = run()
    # Keep the best puzzle so far:
    if sc > top: ans, top = pz, sc
    if time.clock() > tlimit: break
# Print out the best puzzle solution:
print join( [join([ans[(x,y)] for x in rw],'') for y in rh], '\n' )

###################### END ######################
```