Problem. For full problem statement, please go to this page. A summary of the problem is that we need to find any longest possible sequence of 5-letter words (from the *wordle* wordlist) so that for each prefix of the sequence, the last word has positive probability to be the mystery word.

Solution. Our main question is: for each word x, what is the longest length of a guess sequence in which x is the last word?

We will use dynamic programming. Suppose we have already chosen a guess sequence, the state is the remaining valid guess words. We may use a set to represent the state.

Thus the initial state is the entire wordlist. For each valid guess word, we try to use it as a next guess word, and the process terminates if the state becomes an empty set. Suppose we have a state S and a valid guess word x_0, what is the next state? The next state is simply $S \cap S(x_0, x)$, where

$$S(x_0, x) = \left\{ s \in \text{WORDLIST} : (x_0, s, x) \text{ is a valid guess sequence and } s \notin \{x_0, x\} \right\}$$

For the code, we may use a bitset (in c++) or a large number (in python) to represent a set. Lastly, given an integer, how do we find all the set bits? The trick here is to compute the last set bit (LSB) and find which bit it is. Our dynamic programming function looks like this (python):

```
SIZE = 2315    # There are 2315 words in total
TARGET_WORD = x
POW2 = {1<<i : i for i in range(SIZE)}

@functools.lru_cache(None)
def dp(state):

    max_length = 0
    state0 = state

    while state0:
        x0 = POW2[state0 & (-state0)]    # Last Set Bit
        state0 -= state0 & (-state0)

        max_length = max(max_length, dp(state & S(x0, x)))

    return max_length + 1
```

where `S` computes the desired set as said above. You may refer to the `GetMask` function in the source code.

Hence for a target word, this helps us find the maximum length of a guess sequence. To trace back the entire sequence we may use a successor map (you may refer to `dp_save` in the source code)

It takes around approximately 30 seconds to 2 minutes for each target word, so it is pretty slow. The main reason for the slowness is that bit operations for large numbers are quite slow. Note the numbers here are in the range $[1, 2^{2315}]$, thus the numbers may have up to 700 digits. An optimization here is to give priority to target words which have common character structures. For example we would like to spend our resources first to words like *GAMER*, *BOXER*, *BAKER*, *SAFER*... as we believe them to give longer guess sequences. There are many ways to do it,

one of which is to give a score to a word and then sort the wordlist.

There are also many possible ways to score the words. The heuristics here is to counts the occurrences each mask of the word, e.g. for the word *GAMER*, its masks are ■*AMER*, ■*A*■*ER*, *G*■■■*R*, ■■*MER* etc. However, it is not very interesting if we mask too many characters of the word, so in the code it only counts if at most 2 characters are masked.

As a result, optimal answer is found to be using the target words *COWER*, *LOWER*, *MOWER*, *POWER*, *ROWER*, *SOWER* and *TOWER* which has a guess sequence of size 16. These words are the 33-rd, 41-st, 43-rd, 46-th, 49-th, 51-st and 52-nd in the mentioned sorted order, which lies in the first 2% of the entire wordlist. For the complete result, check the `checked.txt`, where it contains for every word its longest guess sequence.