LIJEN

The number of possible words of length k is 2^k. Because words in the dictionary are at most 12 symbols long, the total number of all words that can be used as codewords is 8191. This number is small enough for the following approach:

- Generate all 8191 potential codewords,
- For each codeword P determine the set S of words from the dictionary which differ from P in the least number of symbols,
- If S contains only one word, then P can be uniquely decoded to that word (Q). If the sending time for codeword P is less than the best sending time known for word Q, then set P as the codeword for word Q.

The above gives us the best codeword for each dictionary word.

The time complexity is $O(2^K \cdot N + L)$, where K is the length of the longest word (12).

The official source code represents symbols with bits (instead of strings), which eases the implementation of some parts of the algorithm (for example, the number of symbols in which two words p and q differ is the number of 1 digits in p xor q).

BINGO

For 2×2 and 3×3 cards the problem can be solved in multiple ways because the number of possible cards is small enough. However, there are 16! = 20922789888000 different 4×4 cards and it is not possible (or necessary) to search and score them all in so little time. For simplicity, the remainder of this text assumes the cards are 4×4.

We use a number of optimizations to reduce the number of cards considered:

Idea	**Implementation**
The entire input sequence doesn't matter, just how many times each sequence of 4 numbers appears.	Before starting the search we calculate each possible row's worth (how many times it appears in the input sequence). When searching, we can now score a row in only one operation, instead of going through the entire input sequence.
Swapping two rows does not change the value of the card.	When searching, the order in which we placed the previous rows does not matter, just which of the 16 numbers we've used. There are 2^{16} subsets of 16 numbers. The value of the search function is uniquely determined with the numbers already placed (if placing row by row) so we can cache (memoize) the value for each subset and calculate it at most once.

For any 4 numbers, one of its 24 possible orderings is worth most. If we decide to put the 4 numbers in a single row, then it only makes sense to do it in that, most valuable, ordering.	In an additional step before searching, we calculate for each subset of 4 numbers how much their most valuable ordering is worth. When searching, we place the numbers row by row instead of number by number.
When considering which numbers to put in the next row, one number can be fixed and reduce the number of options. We can do this becuase the ordering of the rows does not matter, and that number will have to be put on the card eventually, so we might as well put it now..	The smallest of the remaining numbers is always placed in the current row, and the remaining 3 are selected in all possible ways.

MRAVOGRAD

Let d(r, c) be the distance from intersection (r, c) to the nearest umbrella.

Let f(r, c) be the set of umbrellas at distance d(r, c) of intersection (r, c). If f(r, c) contains more than one umbrella, then intersection (r, c) is "wet".

To efficiently solve the problem note that, for every rectangle (r1, c1, r2, c2), if f(r1, c1) = f(r1, c2) = f(r2, c1) = f(r2, c2) = X, then f(r, c) = X for every intersection inside the rectangle.

The following divide-and-conquer algorithm will work:

```
for rectangle (r1,c1,r2,c2)
   if f(r1,c1) = f(r1,c2) = f(r2,c1) = f(r2,c2) then
      if f(r1,c1) contains exactly one square
         return 0
      else
         return (r2-r1-1)*(c2-c1-1)
   else
      r = (r1+r2) div 2
      c = (c1+c2) div 2
      recursively solve rectangles
      (r1,c1,r,c), (r1,c+1,r,c2), (r+1,c1,r2,c) i (r+1,c+1,r2,c2)
```