# Documentation

The problem involves finding the maximum product of the lengths of two words in a given list, where the two words share no common letters. The goal is to return the maximum value of length (word[i])×length(word[j]) , or 0 if no such pair exists. Each word in the input consists of lowercase English letters, and the solution needs to handle up to 1000 words efficiently. This makes brute-force solutions that compare every character of two words impractical for large inputs.

To optimize the process, we use bit manipulation, representing each word as a 26-bit integer or "bitmask." Each bit in the bitmask corresponds to a letter from 'a' to 'z', where a bit is set to 1 if the word contains the corresponding letter. For example, the word "abc" is represented as 000...0111000...0111 because 'a', 'b', and 'c' are present. Using bitmasks allows us to check whether two words share letters with a simple bitwise AND operation. If the result is 0, the words have no overlapping letters.

The algorithm begins by precomputing the bitmask and length for each word in the list. This avoids recalculating these values repeatedly during pairwise comparisons. Once these values are prepared, the algorithm iterates through all pairs of words, checking their bitmasks for overlap. If two words have no common letters, their lengths are multiplied, and the maximum product is updated if the new value is higher.

This method is efficient because the bitwise AND operation is very fast, and representing words as integers drastically reduces the complexity of overlap checks. In the worst case, precomputing the bitmasks ensures that the algorithm only needs $O(N^2)$ comparisons, where NN is the number of words. The pre-computation step takes $O(L \cdot N)$, where LL is the average word length.

The algorithm also handles edge cases gracefully. If all words in the list share at least one common letter, the function correctly returns 0. Short words or duplicate words in the list are also managed without impacting the

correctness of the result. The constraints ensure that the input always contains lowercase English letters, making the bitmask approach highly effective.

This solution is a practical demonstration of how bit manipulation can optimize problems involving subsets or uniqueness checks. By reducing the problem to integer operations, it avoids the inefficiencies of character-by-character comparisons. The approach is also versatile and can be applied to other problems that require efficient set operations or overlap checks.

Overall, this problem exemplifies how precomputations and bit manipulation can transform a seemingly complex problem into a manageable and efficient solution. It highlights the power of bitwise operations in scenarios where direct comparisons are costly and showcases their utility in algorithm design for competitive programming and real-world applications.