Maximum Product of Word Length

Given a string array words, return *the maximum value of* length(word[i]) * length(word[j]) *where the two words do not share common letters*. If no such two words exist, return 0.

Example 1:
Input: words = ["abcw","baz","foo","bar","xtfn","abcdef"]
Output: 16
Explanation: The two words can be "abcw", "xtfn".
Example 2:
Input: words = ["a","ab","abc","d","cd","bcd","abcd"]
Output: 4
Explanation: The two words can be "ab", "cd".
Example 3:
Input: words = ["a","aa","aaa","aaaa"]
Output: 0
Explanation: No such pair of words.

Constraints:
- 2 <= words.length <= 1000
- 1 <= words[i].length <= 1000
- words[i] consists only of lowercase English letters.

```
class Solution {


 // 6ms  — passed.
 public int bitNumber(char ch) {
  return (int)ch - (int)'a';
 }

   public int maxProduct(String[] words) {
   int n = words.length;
   int[] masks = new int[n];
   int[] lens = new int[n];
```

```
  int bitmask = 0;
  for (int i = 0; i < n; ++i) {
    bitmask = 0;
    for (char ch : words[i].toCharArray()) {
      // add bit number bit_number in bitmask
       bitmask |= 1 << bitNumber(ch);
    }
    masks[i] = bitmask;
    lens[i] = words[i].length();
  }

  int maxVal = 0;
  for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
      if ((masks[i] & masks[j]) == 0)
        maxVal = Math.max(maxVal, lens[i] * lens[j]);

  return maxVal;
}



/*
  //315ms — passed
  public int maxProduct(String[] words) {
      HashSet<Character> set = new HashSet();
      int i = -1; int j; int k;
      String tmp;
      boolean unique;
      int max = 0;
      int len;

      int wlen = words.length;

      i++;
      for (String w: words){
          set.clear();
          j = -1;
          while (++j < w.length())
              set.add(w.charAt(j));
          j = i;
```

```
            while (++j < wlen){
                tmp = words[j];
                unique = true;
                k = -1;
                while (++k < tmp.length()){
                    if (set.contains(tmp.charAt(k))){
                        unique = false;
                        break;
                    }
                }
                if (unique){
                    len = w.length() * tmp.length() ;
                    max = (max >= len) ? max : len;
                }
            }
        }
        return max;
    }
    */
}
```