Maximum Product of Word Lengths

Given a string array `words`, find the maximum value of `length(word[i]) * length(word[j])` where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1:

Given `["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]`
Return `16`
The two words can be `"abcw", "xtfn"`.

Example 2:

Given `["a", "ab", "abc", "d", "cd", "bcd", "abcd"]`
Return `4`
The two words can be `"ab", "cd"`.

Example 3:

Given `["a", "aa", "aaa", "aaaa"]`
Return `0`
No such pair of words.

Credits:
Special thanks to @dietpepsi for adding this problem and creating all test cases.

Solution 1

```
public class Solution {
    public int maxProduct(String[] words) {
        int max = 0;

        Arrays.sort(words, new Comparator<String>(){
            public int compare(String a, String b){
                return b.length() - a.length();
            }
        });

        int[] masks = new int[words.length]; // alphabet masks

        for(int i = 0; i < masks.length; i++){
            for(char c: words[i].toCharArray()){
                masks[i] |= 1 << (c - 'a');
            }
        }

        for(int i = 0; i < masks.length; i++){
            if(words[i].length() * words[i].length() <= max) break; //prunning
            for(int j = i + 1; j < masks.length; j++){
                if((masks[i] & masks[j]) == 0){
                    max = Math.max(max, words[i].length() * words[j].length());
                    break; //prunning
                }
            }
        }

        return max;
    }
}
```

written by cc9208 original link here

Solution 2

Same algorithm as most, just written a bit shorter.

```
int maxProduct(vector<string>& words) {
    vector<int> mask(words.size());
    int result = 0;
    for (int i=0; i<words.size(); ++i) {
        for (char c : words[i])
            mask[i] |= 1 << (c - 'a');
        for (int j=0; j<i; ++j)
            if (!(mask[i] & mask[j]))
                result = max(result, int(words[i].size() * words[j].size()));
    }
    return result;
}
```

Update: Here's an O(n+N) variation, where N is the total number of characters in all words. Thanks to junhuangli for the suggestion.

```
int maxProduct(vector<string>& words) {
    unordered_map<int,int> maxlen;
    for (string word : words) {
        int mask = 0;
        for (char c : word)
            mask |= 1 << (c - 'a');
        maxlen[mask] = max(maxlen[mask], (int) word.size());
    }
    int result = 0;
    for (auto a : maxlen)
        for (auto b : maxlen)
            if (!(a.first & b.first))
                result = max(result, a.second * b.second);
    return result;
}
```

Or: (thanks to junhuangli's further comment)

```
int maxProduct(vector<string>& words) {
    unordered_map<int,int> maxlen;
    int result = 0;
    for (string word : words) {
        int mask = 0;
        for (char c : word)
            mask |= 1 << (c - 'a');
        maxlen[mask] = max(maxlen[mask], (int) word.size());
        for (auto maskAndLen : maxlen)
            if (!(mask & maskAndLen.first))
                result = max(result, (int) word.size() * maskAndLen.second);
    }
    return result;
}
```

written by StefanPochmann original link here

Solution 3

Sort the vector first according to the length of string. Then use some early pruning to fasten the process.

The worst cases would still be O(N^2). It's faster than most O(N^2) solutions. I don't know whether we can do better. (Binary Search Seems Not work here) Any comments is welcomed.

Update: We can use counting sort to improve the time complexity of sorting to O(N).

```
class Solution {
public:
    int maxProduct(vector<string>& words) {
        int s=words.size();
        if(s==0) return 0;
        vector<int> bit(s,0);
        sort(words.begin(), words.end(), compare);  //sort the vector from longes
t to shortest
        for(int i=0; i<s; i++){ //bit manipulation
            for(int j=0; j<words[i].size(); j++) bit[i] |=(1<<(words[i][j]-'a'));

        }
        int maxlength=0;
        for(int i=0; i<s-1; i++){
            int si=words[i].size();
            if(si*si<=maxlength) break; //early pruning
            for(int j=i+1; j<s; j++){
                int sj=words[j].size();
                if(si*sj<=maxlength) break;  //early pruning
                if((bit[i]&bit[j])==0) maxlength=si*sj;
            }
        }
        return maxlength;
    }
    static bool compare(string a, string b){
        return a.size()>b.size();
    }
};
```

written by zjh08177 original link here