## Longest Word in Dictionary through Deleting

Given a string and a string dictionary, find the longest string in the dictionary that can be formed by deleting some characters of the given string. If there are more than one possible results, return the longest word with the smallest lexicographical order. If there is no possible result, return the empty string.

### Example 1:

```
Input:
s = "abpcplea", d = ["ale","apple","monkey","plea"]

Output:
"apple"
```

### Example 2:

```
Input:
s = "abpcplea", d = ["a","b","c"]

Output:
"a"
```

### Note:

1. All the strings in the input will only contain lower-case letters.
2. The size of the dictionary won't exceed 1,000.
3. The length of all the strings in the input won't exceed 1,000.

## Solution 1

We sort the input dictionary by longest length and lexicography. Then, we iterate through the dictionary exactly once. In the process, the first dictionary word in the sorted dictionary which appears as a subsequence in the input string s must be the desired solution.

```
public String findLongestWord(String s, List<String> d) {
    Collections.sort(d, (a,b) -> a.length() != b.length() ? -Integer.compare(a.le
ngth(), b.length()) :  a.compareTo(b));
    for (String dictWord : d) {
        int i = 0;
        for (char c : s.toCharArray())
            if (i < dictWord.length() && c == dictWord.charAt(i)) i++;
        if (i == dictWord.length()) return dictWord;
    }
    return "";
}
```

An alternate, more efficient solution which avoids sorting the dictionary:

```
public String findLongestWord(String s, List<String> d) {
    String longest = "";
    for (String dictWord : d) {
        int i = 0;
        for (char c : s.toCharArray())
            if (i < dictWord.length() && c == dictWord.charAt(i)) i++;

        if (i == dictWord.length() && dictWord.length() >= longest.length())
            if (dictWord.length() > longest.length() || dictWord.compareTo(longes
t) < 0)
                longest = dictWord;
    }
    return longest;
}
```

Time Complexity: O(nk), where n is the length of string s and k is the number of words in the dictionary.

written by compton_scatter original link here

## Solution 2

```
def findLongestWord(self, s, d):
    def isSubsequence(x):
        it = iter(s)
        return all(c in it for c in x)
    return max(sorted(filter(isSubsequence, d)) + [''], key=len)
```

More efficient version (no sorting):

```
def findLongestWord(self, s, d):
    def isSubsequence(x):
        it = iter(s)
        return all(c in it for c in x)
    return min(filter(isSubsequence, d) + [''], key=lambda x: (-len(x), x))
```

Different style:

```
def findLongestWord(self, s, d):
    best = ''
    for x in d:
        if (-len(x), x) < (-len(best), best):
            it = iter(s)
            if all(c in it for c in x):
                best = x
    return best
```

Optimized as suggested by @easton042, testing from longest to shortest and returning the first valid one without testing the rest:

```
def findLongestWord(self, s, d):
    def isSubsequence(x):
        it = iter(s)
        return all(c in it for c in x)
    d.sort(key=lambda x: (-len(x), x))
    return next(itertools.ifilter(isSubsequence, d), '')
```

Or:

```
def findLongestWord(self, s, d):
    for x in sorted(d, key=lambda x: (-len(x), x)):
        it = iter(s)
        if all(c in it for c in x):
            return x
    return ''
```

And taking that even further by not sorting unnecessarily much:

```
def findLongestWord(self, s, d):
    heap = [(-len(word), word) for word in d]
    heapq.heapify(heap)
    while heap:
        word = heapq.heappop(heap)[1]
        it = iter(s)
        if all(c in it for c in word):
            return word
    return ''
```

written by StefanPochmann original link here

## Solution 3

I think there is no need to sort the dic, just iterate the dic and test whether the word is satisfied and whether we need update our answer.

```
    string findLongestWord(string s, vector<string>& d) {
        string ans;
        for (int i = 0; i < d.size(); i++) {
            int pi = 0, pj = 0;
            for (; pi < s.size() && pj < d[i].size(); pi++) {
                pj += s[pi] == d[i][pj];
            }
            if (pj == d[i].size() && (ans.size() < d[i].size() || (ans.size() ==
d[i].size() && ans > d[i])))
                ans = d[i];
        }
        return ans;
    }
```

written by saxion original link here