691. Stickers to Spell Word

DescriptionHintsSubmissionsDiscussSolution

- Difficulty:Hard
- Total Accepted:1.6K
- Total Submissions:5K
- Contributor: 1337c0d3r

We are given N different types of stickers. Each sticker has a lowercase English word on it.

You would like to spell out the given `target` string by cutting individual letters from your collection of stickers and rearranging them.

You can use each sticker more than once if you want, and you have infinite quantities of each sticker.

What is the minimum number of stickers that you need to spell out the `target`? If the task is impossible, return -1.

Example 1:

Input:

```
["with", "example", "science"], "thehat"
```

Output:

```
3
```

Explanation:

```
We can use 2 "with" stickers, and 1 "example" sticker.
After cutting and rearrange the letters of those stickers, we can form the
target "thehat".
Also, this is the minimum number of stickers necessary to form the target
string.
```

Example 2:

Input:

```
["notice", "possible"], "basicbasic"
```

Output:

```
-1
```

Explanation:

```
We can't form the target "basicbasic" from cutting letters from the given
stickers.
```

Note:

- `stickers` has length in the range `[1, 50]`.
- `stickers` consists of lowercase English words (without apostrophes).
- `target` has length in the range `[1, 15]`, and consists of lowercase English letters.
- In all test cases, all words were chosen randomly from the 1000 most common US English words, and the target was chosen as a concatenation of two random words.
- The time limit may be more challenging than usual. It is expected that a 50 sticker test case can be solved within 35ms on average.

There are potentially a lot of overlapping sub problems, but meanwhile we don't exactly know what those sub problems are. DP with memoization works pretty well in such cases. The workflow is like backtracking, but with memoization. Here I simply use a sorted string of target as the key for the unordered_map DP. A sorted target results in a unique sub problem for possibly different strings.

```
dp[s] is the minimum stickers required for string s (-1 if impossible). Note s
is sorted.
clearly, dp[""] = 0, and the problem asks for dp[target].
```

The DP formula is:

```
dp[s] = min(1+dp[reduced_s]) for all stickers,
here reduced_s is a new string after certain sticker applied
```

Optimization: If the target can be spelled out by a group of stickers, at least one of them has to contain character target[0]. So I explicitly require next sticker containing target[0], which significantly reduced the search space.

BTW, I am not good at Java. Looking forward to your revision!

```
class Solution {
public:
    int minStickers(vector<string>& stickers, string target) {
        int m = stickers.size();
        vector<vector<int>> mp(m, vector<int>(26, 0));
        unordered_map<string, int> dp;
        // count characters a-z for each sticker
        for (int i = 0; i < m; i++)
            for (char c:stickers[i]) mp[i][c-'a']++;
        dp[""] = 0;
        return helper(dp, mp, target);
    }
private:
    int helper(unordered_map<string, int>& dp, vector<vector<int>>& mp, string
target) {
        if (dp.count(target)) return dp[target];
        int ans = INT_MAX, n = mp.size();
        vector<int> tar(26, 0);
        for (char c:target) tar[c-'a']++;
        // try every sticker
        for (int i = 0; i < n; i++) {
```

```
            // optimization
            if (mp[i][target[0]-'a'] == 0) continue;
            string s;
            // apply a sticker on every character a-z
            for (int j = 0; j < 26; j++)
                if (tar[j]-mp[i][j] > 0) s += string(tar[j]-mp[i][j], 'a'+j);
            int tmp = helper(dp, mp, s);
            if (tmp!= -1) ans = min(ans, 1+tmp);
        }
        dp[target] = ans == INT_MAX? -1:ans;
        return dp[target];
    }
};

class Solution(object):
    def minStickers(self, stickers, target):
        m = len(stickers)
        mp = [[0]*26 for y in range(m)]
        for i in range(m):
            for c in stickers[i]:
                mp[i][ord(c)-ord('a')] += 1
        dp = {}
        dp[""] = 0

        def helper(dp, mp, target):
            if target in dp:
                return dp[target]
            n = len(mp)
            tar = [0]*26
            for c in target:
                tar[ord(c)-ord('a')] += 1
            ans = sys.maxint
            for i in xrange(n):
                if mp[i][ord(target[0])-ord('a')] == 0:
                    continue
                s = ''
                for j in range(26):
                    if tar[j] > mp[i][j]:
                        s += chr(ord('a')+j)*(tar[j] - mp[i][j])
                tmp = helper(dp, mp, s)
                if (tmp != -1):
                    ans = min(ans, 1+tmp)
            dp[target] = -1 if ans == sys.maxint else ans
            return dp[target]

        return helper(dp, mp, target)
```