514. Freedom Trail

Add to List

Description SubmissionsSolutions

- Total Accepted: **1726**
- Total Submissions: **5196**
- Difficulty: **Hard**
- Contributors: **Rabby250**

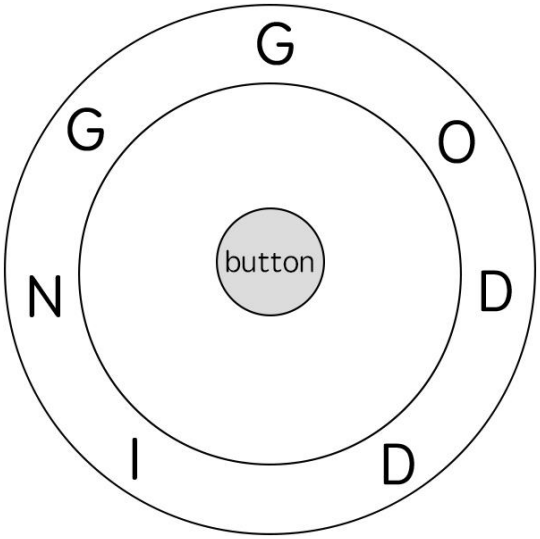
In the video game Fallout 4, the quest "Road to Freedom" requires players to reach a metal dial called the "Freedom Trail Ring", and use the dial to spell a specific keyword in order to open the door.

Given a string **ring**, which represents the code engraved on the outer ring and another string **key**, which represents the keyword needs to be spelled. You need to find the **minimum** number of steps in order to spell all the characters in the keyword.

Initially, the first character of the **ring** is aligned at 12:00 direction. You need to spell all the characters in the string **key** one by one by rotating the ring clockwise or anticlockwise to make each character of the string **key** aligned at 12:00 direction and then by pressing the center button.
At the stage of rotating the ring to spell the key character **key[i]**:

1. You can rotate the **ring** clockwise or anticlockwise **one place**, which counts as 1 step. The final purpose of the rotation is to align one of the string **ring's** characters at the 12:00 direction, where this character must equal to the character **key[i]**.
2. If the character **key[i]** has been aligned at the 12:00 direction, you need to press the center button to spell, which also counts as 1 step. After the pressing, you could begin to spell the next character in the key (next stage), otherwise, you've finished all the spelling.

Example:




```
Input: ring = "godding", key = "gd"

Output: 4

Explanation:
 For the first key character 'g', since it is already in place, we just need
 1 step to spell this character.
 For the second key character 'd', we need to rotate the ring "godding" anti
clockwise by two steps to make it become "ddinggo".
 Also, we need 1 more step for spelling.
 So the final output is 4.
```

Note:

1. Length of both ring and **key** will be in range 1 to 100.
2. There are only lowercase letters in both strings and might be some duplcate characters in both strings.
3. It's guaranteed that string **key** could always be spelled by rotating the string **ring**.

Subscribe to see which companies asked this question.

Solution:

1. recording each index of characters in ring,beacuse each characters we have search in this time would be starting index in the next search

2. How could we solve the problem that rotate the ring to the left or right ? My idea is **min((tmp[j] + size -it)%size,(it + size - tmp[j])%size)**

- Suppose you want to rotate the ring to the right and search 'k', and the size is 5. We could calculate it by **this + size -k(index)%size**

```
this -  -  -  -  k
```

- If we want to rotate the ring to the left，what should we do? It is the same problem with above problem,move **this** to it 's right,and reach **k**

```
k -  -  -  -   this
```

- So we could calculate it by **k(index) + size -this%size**
- There are many people use **abs()** instead of **%size**,I think it's faster than mine :)

```
class Solution {
public:
    int findRotateSteps(string ring, string key) {
        int ksize = key.size();
        int size = ring.size();
        //stored index of each characters in ring,pay attention to duplcate
characters.
        unordered_map<char,vector<int>> mp;
        for(int i=0;i<size;++i){
            mp[ring[i]].push_back(i);
        }
        // TODO(zzw):
        // initializing dp vector dp(ksize,size)
        // stores the minimum distance between key[ksize] to rings[size]
        vector<vector<int>> dp(ksize+1,vector<int> (size,INT_MAX));
        fill(dp[0].begin(),dp[0].end(),0);
        vector<int> tmp(1,0);
        int res = INT_MAX;
        for(int i=1;i<=ksize;i++)
        {
            for(auto it: mp[key[i-1]])
            {
                for(int j=0;j<tmp.size();j++)
                {
                    int                     minDist                     =
dp[i-1][tmp[j]]+min((tmp[j]+size-it)%size,(it+size-tmp[j])%size);
```

```
                    dp[i][it] = min(dp[i][it],minDist);
                    res = i!=ksize?res:min(res,minDist);
                }
            }
            tmp = mp[key[i-1]];
        }
        return res + ksize;
    }
};
```