> 输入一个字符串ring代表圆盘上的字符（指针位置在 12 点钟方向，初始指向ring[0]），再输入一个字符串key代表你需要拨动圆盘输入的字符串，你的算法需要返回输入这个key至少进行多少次操作（拨动一格圆盘和按下圆盘中间的按钮都算是一次操作）。

分析： 状态：当前需要输入的字符和当前圆盘指针的位置

> 「状态」就是i和j两个变量。我们可以用i表示当前圆盘上指针指向的字符（也就是ring[i]）；用j表示需要输入的字符（也就是key[j]）

选择：如何拨动圆盘上的指针指向要输入字符

> 对于现在想输入的字符key[j]，我们可以如何拨动圆盘，得到这个字符?

代码框架：

```
int dp(std::string s, int i, std::string t, int j) {
    if(j == t.size()) {
        return 0;
    }

    // 做选择
    int res = INT_MAX;
    for(int k : 字符key[j]在字符串ring中所有索引位置){
        res = min(把i顺时针拨到k的代价，把i逆时针拨到k的代价);
    }

    return res;
}
```

解法如下：

```
class Solution {
public:
  int findRotateSteps(std::string ring, std::string key) {
    int m = ring.size();
    int n = key.size();
    // 备忘录初始化为0
    memo.resize(m + 1, std::vector<int>(n + 1, 0));

    //记录圆环上字符索引位置
    for (int i = 0; i < m; i++) {
      charToIndex[ring[i]].push_back(i);
```

```
    }

    // 圆盘最初指向12点钟方向，从0开始
    return dp(ring, 0, key, 0);
  }

private:
  int dp(std::string s, int indexs, std::string key, int indexk) {
    if (indexk == key.size()) {
      return 0;
    }

    if (memo[indexs][indexk] != 0) {
      return memo[indexs][indexk];
    }

    int n = s.size();
    // 做选择
    int res = INT_MAX;

    for (auto k : charToIndex[key[indexk]]) {
      // 转动指针次数
      int delta = abs(k - indexs);
      // 选择顺时针或逆时针
      delta = std::min(delta, n - delta);
      // 子问题求解
      int subProblem = dp(s, k, key, indexk + 1);

      // 整体求解
      res = std::min(res, 1 + delta + subProblem);
    }

    // 结果加入备忘录
    memo[indexs][indexk] = res;

    return res;
  }

  std::unordered_map<char, std::vector<int>> charToIndex;
  std::vector<std::vector<int>>              memo;
};
```