查阅更多的题解，请点击

Problem

91. Decode Ways(Medium)

A message containing letters from A-Z is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given a non-empty string containing only digits, determine the total number of ways to decode it.

Example 1:

```
Input: "12"
Output: 2
Explanation: It could be decoded as "AB" (1 2) or "L" (12).
```

Example 2:

```
Input: "226"
Output: 3
Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
```

Solution

Corner Cases：

- 当前需要解析的字符为‘0’
  - 若该字符为字符串第一个字符，解码失败
  - 该字符不为首字符，前一个字符不为'1'，解码失败

O(n) time, O(1) space

输入可以看做是字符数组，还是常规方法，观察原问题是否可以reduce。

这里可以将较大的原问题reduce成较小的子问题：前k个字符，可以有几种解码方式，设为$dp[k-1]$，令s_{k-1}代表第i个字符，有以下的递推关系式：

$$dp[k] = \begin{cases} 解析失败 & s_k = 0 \ and \ s_{k-1} \neq 1 \\ dp[k-2] & s_k = 0 \ and \ s_{k-1} = 1 \\ dp[k-2] + dp[k-1] & s_{k-1} = 1 \ or \ (s_{k-1} = 2 \ and \ s_k <= 6) \\ dp[k-2] & else \end{cases}$$

可以发现是一个典型的**动态规划**问题

```
class Solution
{
  private:
    bool isValid(char first, char second)
    {
        return first == '1' || (first == '2' && second <= '6');
    }

  public:
    int numDecodings(string s)
    {
        if (s.empty() || s[0] == '0')
            return 0;
        vector<int> dp(s.size(), 0);
        dp[0] = 1;
        if (s.size() == 1)
            return dp[0];
        if (s[1] == '0')
        {
            if (!isValid(s[0], s[1]))
                return 0;
            dp[1] = 1;
        }
        else
        {
            if (isValid(s[0], s[1]))
                dp[1] = 2;
            else
                dp[1] = 1;
        }
        if (s.size() == 2)
            return dp[1];
        for (int i = 2; i < s.size(); ++i)
        {
            if (s[i] == '0')
            {
                if (!isValid(s[i - 1], s[i]))
                    return 0;
                dp[i] = dp[i - 2];
            }
            else
            {
                if (isValid(s[i - 1], s[i]))
                    dp[i] = dp[i - 1] + dp[i - 2];
                else
                    dp[i] = dp[i - 1];
            }
        }
        return dp[s.size() - 1];
    }
};
```

注意上述算法是O(n) space的，然而在更新过程中仅涉及$dp[k], dp[k-1], dp[k-2]$，可以做下述优化，得到O(1) space的解法

```
class Solution
{
  private:
    bool isValid(char first, char second)
    {
        return first == '1' || (first == '2' && second <= '6');
    }

  public:
    int numDecodings(string s)
    {
        if (s.empty() || s[0] == '0')
            return 0;
        if (s.size() == 1)
            return 1;
        int cur = 0, f_1 = 1, f_2 = 1;
        for (int i = 1; i < s.size(); ++i)
        {
            int temp = f_1;
            if (s[i] == '0')
            {
                if (!isValid(s[i - 1], s[i]))
                    return 0;
                cur = f_2;
            }
            else
            {
                if (isValid(s[i - 1], s[i]))
                    cur = f_1 + f_2;
                else
                    cur = f_1;
            }
            f_1 = cur;
            f_2 = temp;
        }
        return cur;
    }
};
```

GitHub传送门