查阅更多的题解，请点击

# Problem

## 279. Perfect Squares(Medium)

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

**Example 1:**

```
Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.
```

**Example 2:**

```
Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.
```

# Solution

## DP算法

**O(n) time, O(n) space**

观察问题，考虑是否可以reduce原问题，一个直观的子问题：对于$k(k < n)$，符合题意的解为多少?

子问题具有重叠，考虑**动态规划**。很难直观得出递归表达式，这里从最简单的问题出发，观察规律：

- n=1, result={1}=1
- n=2, result={1,1}=2
- n=3, result={1,1,1}=3
- n=4, result={4}=1
- n=5, result={4,1}=2
- ……

对于n，分解出的因子都是square number，那么根据square number可以得出以下的递归表达式：

$$DP(k) = min(\{DP[i*i] + DP[k - i*i], i = 1, 2, 3, \cdots \ and \ i*i <= k\})$$

注意$DP[i*i] = 1$

```
class Solution
{
public:
    int numSquares(int n)
    {
        if (n < 1)
            return 0;
        vector<int> dp(n + 1, INT_MAX);
        dp[0] = 0;
        for (int i = 1; i <= n; ++i)
        {
            for (int j = 1; j * j <= i; ++j)
            {
                dp[i] = min(dp[i], dp[i - j * j] + 1);
            }
        }
        return dp[n];
    }
};
```

上述算法可以得出解，注意其中关键的**dp数组**，其中dp[k]的值是固定的，所以如果多次重复测试，可以保留之前计算的dp，这里可以使用静态变量完成：

```
class Solution
{
public:
    int numSquares(int n)
    {
        if (n < 1)
            return 0;
        static vector<int> dp = {0};
        while (dp.size() <= n)
        {
            int cur = INT_MAX;
            int i = dp.size();
            for (int j = 1; j * j <= i; ++j)
            {
                cur = min(cur, dp[i - j * j] + 1);
            }
            dp.emplace_back(cur);
        }
        return dp[n];
    }
};
```

# 四平方和定理

四平方和定理说明每个正整数均可表示为4个整数的平方和，那么这道题的结果只可能是1,2,3,4中的一个：

- 当n=i*i时，结果为1

- 当n=(4^ a)*(8b+7)时，结果为4
- 剩余两种情况自行讨论

```
class Solution
{
private:
    int is_square(int n)
    {
        int sqrt_n = (int)(sqrt(n));
        return (sqrt_n*sqrt_n == n);
    }

public:
    int numSquares(int n)
    {
        if(is_square(n))
        {
            return 1;
        }

        while ((n & 3) == 0)
        {
            n >>= 2;
        }
        if ((n & 7) == 7)
        {
            return 4;
        }

        int sqrt_n = (int)(sqrt(n));
        for(int i = 1; i <= sqrt_n; i++)
        {
            if (is_square(n - i*i))
            {
                return 2;
            }
        }

        return 3;
    }
};
```

### Corner Cases：

GitHub传送门