完全背包：279. 完全平方数

问题描述

给定一个正整数 n，你需要找到最少数量的完全平方数（例如 1, 4, 9, 16, ...），使它们的和等于 n。

代码分析

1. 变量与初始化

```cpp
int m = sqrt(n);
vector<vector<int>> dp(m + 1, vector<int>(n + 1));
for(int j = 1; j <= n; j++) dp[0][j] = 0x3f3f3f3f;
```

- m = sqrt(n)：我们首先计算 n 的平方根 m，因为最大的完全平方数是 m^2，我们只需要考虑从 1^2 到 m^2 的完全平方数即可。
- dp 数组：定义一个二维动态规划数组 dp[i][j]，表示前 i 个完全平方数凑成数字 j 的最少数量的完全平方数。
- 初始化 dp[0][j]：表示不使用任何平方数时无法凑成 j，因此将其初始化为无穷大（代码中使用 0x3f3f3f3f 作为无穷大）。

2. 动态规划填表

```cpp
for (int i = 1; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        dp[i][j] = dp[i - 1][j];  // 不选第 i 个完全平方数
        if (j >= i * i) {
            dp[i][j] = min(dp[i][j], dp[i][j - i * i] + 1);  // 选第 i 个完全平方数
        }
    }
}
```

- **外层循环** i：从 1 遍历到 m，表示正在考虑第 i 个完全平方数（i^2）。
- **内层循环** j：从 0 遍历到 n，表示我们要构成的数字是 j。
- dp[i][j] = dp[i - 1][j]：表示不使用当前平方数 i^2 的最优解。
- if (j >= i*i)：表示当前数字 j 大于等于 i^2 时，可以尝试使用当前平方数 i^2。
- dp[i][j] = min(dp[i][j], dp[i][j - i * i] + 1)：如果我们使用了平方数 i^2，则从 dp[i][j - i * i] 状态转移过来，并且需要再加上 1（表示使用了一个平方数）。

3. 返回结果

```cpp
return dp[m][n];
```

最终返回 dp[m][n]，它表示用最少的完全平方数凑成 n 的方案数。

举例说明

假设 n = 12：

- 最大的完全平方数是 m = sqrt(12) ≈ 3，即 1^2, 2^2, 3^2，我们只需要考虑这三个完全平方数。
- 我们需要通过这三个平方数凑出 12，找出最少的组合数。

动态规划表构建过程

初始化

- dp[0][1] 到 dp[0][12] 都是无穷大，因为无法用 0 个完全平方数凑成任何正数。
- dp[i][0] = 0，因为用 0 个平方数可以凑成 0。

逐步填表

1. **第一轮 (i = 1, 考虑 1^2)**：
   - dp[1][j] 表示用 1^2 来凑成 j。
   - 可以直接构造出 dp[1][1] = 1, dp[1][2] = 2, ……, dp[1][12] = 12。
   - 状态转移公式：dp[1][j] = dp[1][j - 1^2] + 1。
   - dp[1] 数组更新为：

```plaintext
dp[1] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

2. **第二轮 (`i = 2`, 考虑 `2^2`)**:
   - `dp[2][j]` 表示用 `1^2` 和 `2^2` 来凑成 `j`。
   - `dp[2][4] = 1`（用 `2^2` 直接构造 4）。
   - `dp[2][5] = 2`（`1^2 + 2^2`）。
   - `dp[2][6] = 3`（`1^2 + 1^2 + 2^2`）。
   - 继续构造直到 `dp[2][12]`，结果更新为：

```plaintext
dp[2] = [0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3]
```

3. **第三轮 (`i = 3`, 考虑 `3^2`)**:
   - `dp[3][9] = 1`，因为用 `3^2` 可以直接构造 9。
   - `dp[3][10] = 2`（`1^2 + 3^2`）。
   - `dp[3][11] = 3`（`1^2 + 1^2 + 3^2`）。
   - `dp[3][12] = 3`（`3^2 + 2^2`）。

最终的 dp 数组结果：

```plaintext
dp[3] = [0, 1, 2, 3, 1, 2, 3, 4, 2, 1, 2, 3, 3]
```

因此，`dp[3][12] = 3`，表示最少使用 3 个完全平方数可以构造 12。

- 最优组合是：`4 + 4 + 4`（即 `2^2 + 2^2 + 2^2`）。

代码的时间复杂度

- 外层循环运行 m 次，其中 `m = sqrt(n)`。
- 内层循环运行 n 次。
- 总时间复杂度为 **O(m * n)**，即 **O(sqrt(n) * n)**。

代码优化建议

这段代码使用了二维的 dp 数组，可以优化为**一维动态规划**，减少空间复杂度到 `O(n)`。你只需要维护一个 `dp[j]` 数组，表示构造 j 的最少完全平方数的个数即可。

优化后的一维 DP 代码：

```cpp
class Solution {
public:
    int numSquares(int n) {
        vector<int> dp(n + 1, INT_MAX);
        dp[0] = 0;

        for (int i = 1; i * i <= n; i++) {
            for (int j = i * i; j <= n; j++) {
                dp[j] = min(dp[j], dp[j - i * i] + 1);
            }
        }
        return dp[n];
    }
};
```

```
//优化前
class Solution {
public:
    int numSquares(int n) {
        int m = sqrt(n);
        vector<vector<int>> dp(m + 1, vector<int>(n + 1));
        for(int j =1;j <= n;j++) dp[0][j] = 0x3f3f3f3f;
        for (int i = 1; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                dp[i][j] = dp[i - 1][j];
                if (j >= i*i) dp[i][j] = min(dp[i][j],dp[i][j-i*i] + 1);
            }
        }
        return dp[m][n];
    }
};

//优化后
class Solution {
public:
    int numSquares(int n) {
        int m = sqrt(n);
        vector<int> dp(n + 1,0x3f3f3f3f);
        dp[0] = 0;
        for (int i = 1; i <= m; i++) {
            for (int j = i*i;j <= n ; j++) {
               dp[j] = min(dp[j],dp[j-i*i] + 1);
            }
        }
        return dp[n];
    }
};
```