# 题目

给你一个整数 n ，返回 和为 n 的完全平方数的最少数量 。完全平方数 是一个整数，其值等于另一个整数的平方；换句话说，其值等于一个整数自乘的积。例如，1、4、9 和 16 都是完全平方数，而 3 和 11 不是。示例 1：输入：n = 12输出：3 解释：12 = 4 + 4 + 4示例 2：输入：n = 13输出：2解释：13 = 4 + 9 提示：1 <= n <= 10^4

# 分析

## 方法一：贪心

思路：

可以使用贪心算法和广度优先搜索（BFS）来解决这个问题。具体做法是，首先生成一个小于等于n的所有完全平方数的列表，然后使用BFS来找到和为n的最少完全平方数数量。

class Solution {

public:

int numSquares(int n) {

// 生成小于等于n的所有完全平方数的列表

vector<int> squares;

for (int i = 1; i \* i <= n; ++i) {

squares.push\_back(i \* i);

}

// 使用BFS找到和为n的最少完全平方数数量

queue<int> q;

unordered\_set<int> visited;

q.push(n);

visited.insert(n);

int level = 0;

while (!q.empty()) {

int size = q.size();

level++;

for (int i = 0; i < size; ++i) {

int num = q.front();

q.pop();

for (int square : squares) {

int diff = num - square;

if (diff == 0) {

return level;

} else if (diff < 0) {

break;

}

if (visited.find(diff) == visited.end()) {

q.push(diff);

visited.insert(diff);

}

}

}

}

return 0;

}

};

这段代码首先生成小于等于n的所有完全平方数的列表vector<int> squares，然后使用BFS来找到和为n的最少完全平方数数量。

## 方法二：动态规划

思路：

你可以使用动态规划来解决这个问题。具体做法是，定义一个数组dp，其中dp[i]表示和为i的完全平方数的最少数量。初始时，将dp数组全部初始化为INT\_MAX，然后将dp[0]初始化为0。接下来，对于每个i，从1遍历到sqrt(i)，计算i-j\*j的值，并更新dp[i]为min(dp[i], dp[i-j\*j]+1)。最终，dp[n]即为所求结果。

代码：

class Solution {

public:

int numSquares(int n) {

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];

}

};