0279. 完全平方数

ITCharge 大约 4 分钟

- 标签：广度优先搜索、数学、动态规划
- 难度：中等

题目链接

- 0279. 完全平方数 - 力扣

题目大意

描述：给定一个正整数 n。从中找到若干个完全平方数（比如 $1, 4, 9, 16 \ldots$），使得它们的和等于 n。

要求：返回和为 n 的完全平方数的最小数量。

说明：

- $1 \le n \le 10^4$。

示例：

- 示例 1：

```py
输入：n = 12
输出：3 
解释：12 = 4 + 4 + 4
```

- 示例 2：

```py
输入：n = 13
输出：2
解释：13 = 4 + 9
```

解题思路

暴力枚举思路：对于小于 n 的完全平方数，直接暴力枚举所有可能的组合，并且找到平方数个数最小的一个。

并且对于所有小于 n 的完全平方数（$k = 1, 4, 9, 16, ...$），存在公式：$ans(n) = min(ans(n - k) + 1), k = 1, 4, 9, 16...$

即： **n 的完全平方数的最小数量 == n - k 的完全平方数的最小数量 + 1**。

我们可以使用递归解决这个问题。但是因为重复计算了中间解，会产生堆栈溢出。

那怎么解决重复计算问题和避免堆栈溢出？

我们可以转换一下思维。

1. 将 n 作为根节点，构建一棵多叉数。
2. 从 n 节点出发，如果一个小于 n 的数刚好与 n 相差一个平方数，则以该数为值构造一个节点，与 n 相连。

那么求解和为 n 的完全平方数的最小 就变成了求解这棵树从根节点 n 到节点 0 的最短路径，或者说树的最小深度。

这个过程可以通过广度优先搜索来做。

思路 1：广度优先搜索

1. 定义 $visited$ 为标记访问节点的 set 集合变量，避免重复计算。定义 $queue$ 为存放节点的队列。使用 $count$ 表示为树的最小深度，也就是和为 n 的完全平方数的最小数量。
2. 首先，我们将 n 标记为已访问，即 `visited.add(n)`。并将其加入队列 $queue$ 中，即 `queue.append(n)`。
3. 令 $count$ 加 1，表示最小深度加 1。然后依次将队列中的节点值取出。
4. 对于取出的节点值 $value$，遍历可能出现的平方数（即遍历 $[1, \sqrt{value} + 1]$ 中的数）。
5. 每次从当前节点值减去一个平方数，并将减完的数加入队列。
    1. 如果此时的数等于 0，则满足题意，返回当前树的最小深度。
    2. 如果此时的数不等于 0，则将其加入队列，继续查找。

思路 1：代码

```py
class Solution:
    def numSquares(self, n: int) -> int:
        if n == 0:
            return 0

        visited = set()
        queue = collections.deque([])

        visited.add(n)
        queue.append(n)

        count = 0
        while queue:
            // 最少步数
            count += 1
            size = len(queue)
            for _ in range(size)·
                value = queue.po
                for i in range(1, int(math.sqrt(value)) + 1):
                    x = value - i * i
                    if x == 0:
                        return count
                    if x not in visited:
                        queue.appendleft(x)
                        visited.add(x)
```

思路 1：复杂度分析

- **时间复杂度**：$O(n \times \sqrt{n})$。
- **空间复杂度**：$O(n)$。

思路 2：动态规划

我们可以将这道题转换为「完全背包问题」中恰好装满背包的方案数问题。

1. 将 $k = 1, 4, 9, 16, ...$ 看做是 k 种物品，每种物品都可以无限次使用。
2. 将 n 看做是背包的装载上限。
3. 这道题就变成了，从 k 种物品中选择一些物品，装入装载上限为 n 的背包中，恰好装满背包最少需要多少件物品。

1. 划分阶段

按照当前背包的载重上限进行阶段划分。

2. 定义状态

定义状态 $dp[w]$ 表示为：从完全平方数中挑选一些数，使其和恰好凑成 w ，最少需要多少个完全平方数。

3. 状态转移方程

$dp[w] = min\{dp[w], dp[w - num] + 1$

4. 初始条件

- 恰好凑成和为 0，最少需要 0 个完 方数。
- 默认情况下，在不使用完全平方数时，都不能恰好凑成和为 w ，此时将状态值设置为一个极大值（比如 $n + 1$），表示无法凑成。

5. 最终结果

根据我们之前定义的状态，$dp[w]$ 表示为：将物品装入装载上限为 w 的背包中，恰好装满背包，最少需要多少件物品。 所以最终结果为 $dp[n]$。

1. 如果 $dp[n] \ne n + 1$，则说明：$dp[n]$ 为装入装载上限为 n 的背包，恰好装满背包，最少需要的物品数量，则返回 $dp[n]$。
2. 如果 $dp[n] = n + 1$，则说明：无法恰好装满背包，则返回 -1。因为 n 肯定能由 n 个 1 组成，所以这种情况并不会出现。

思路 2：代码

```py
class Solution:
    def numSquares(self, n: int) -> int:
        dp = [n + 1 for _ in range(n + 1)]
        dp[0] = 0

        for i in range(1, int(sqrt(n)) + 1):
            num = i * i
            for w in range(num, n + 1):
                dp[w] = min(dp[w], dp[w - num] + 1)

        if dp[n] != n + 1:
            return dp[n]
        return -1
```

思路 2：复杂度分析

- **时间复杂度**：$O(n \times \sqrt{n})$。
- **空间复杂度**：$O(n)$。