Leetcode刷题笔记

目录

[Leetcode刷题笔记 1](#_Toc128841945)

[1. No0279 1](#_Toc128841946)

[1.1 问题描述 1](#_Toc128841947)

[1.2 解题分析 2](#_Toc128841948)

[1.3 优化 3](#_Toc128841949)

[2. No0982：按位与为0的三元组 4](#_Toc128841950)

[2.1 解法1：双二重循环 4](#_Toc128841951)

[2.2 解法2：枚举+子集优化 4](#_Toc128841952)

[3. BFS解题思路 4](#_Toc128841953)

**[Notations and Abbreviations]**

# No0279

## 问题描述

给定正整数 n，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n。你需要让组成和的完全平方数（可以重复）的个数最少。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

## 解题分析

本题可以转化为图搜索中的最短路径问题，因此可以用广度优先搜索算法来解决。

举个例子，令n = 30, m=floor(sqrt(n)) = 5. 则构造n的完全平方数的和，可以使用模块（即数字）为1~5。从中任选一个k使用得到(n=30)的邻节点(n-k\*k)。针对每个节点都执行同样操作，直到最后到达值为0的节点。如下图所示：



Edge上的数字表示要用于构成完全平方数和的数（其实是其平方），对应的子节点的值等于父节点的值减去该数的平方。

广度优先搜索的三个基本要素：

1. 队列管理
2. 节点的表示以及邻节点的遍历
3. 已访问节点的管理

算法流程如下(python-style, general flow for BFS)：

初始化：

创建Queue, visited对象

Add start node (n) to Queue, together with its layer (for start node, it should be zero)

Add start node to visited

While (Queue is not empty):

value, layer = Queue.pop()

if value is the target (it is 0 in this problem):

return layer+1

Traverse all the neighbour nodes, for each of them:

If neighbourNode is not in visited:

Add neighbourNode to visited:

Queue.push(neighbourNode)

在本问题中， 在本问题中，邻节点即从当前节点出发，将每个可用的数（的平方）用一次（即减去）后能到达的下一个值。可以以递推关系式表示如下：



## 优化

四平方定理： 任何一个正整数都可以表示成不超过四个整数的平方之和。 推论：满足四数平方和定理的数n（四个整数的情况），必定满足 n=4^a(8b+7)

这道题如果知道数学定理之后，相当于告诉你：

任何正整数都可以拆分成不超过4个数的平方和 ---> 答案只可能是1,2,3,4

如果一个数最少可以拆成4个数的平方和，则这个数还满足 n = (4^a)\*(8b+7) ---> 因此可以先看这个数是否满足上述公式，如果不满足，答案就是1,2,3了

如果这个数本来就是某个数的平方，那么答案就是1，否则答案就只剩2,3了

如果答案是2，即n=a^2+b^2，那么我们可以枚举a，来验证，如果验证通过则答案是2

只能是3

# No0982：按位与为0的三元组

暴力破解的话，复杂度为O(n^3)。

## 解法1：双二重循环

分而治之。

先进行二重循环，并统计出现的每种结果的个数，存储于一个数组（或哈希表）A中。数组大小取决于输入数的范围。本题中数据范围定义为[0,2\*\*16)，因此该数组大小为最大2\*\*16。

然后再进行数组A与nums之间相与的二重循环。

这样，复杂度降为O(n^2+2^16\*n)~ O(n^2)。

## 解法2：枚举+子集优化



# BFS解题思路

BFS 其实是很简单的基础算法，抓住如下几点即可轻松写出不易错的 baseline:

BFS 算法组成的 3 元素：队列，入队出队的节点，已访问的集合。

队列：先入先出的容器；

节点：最好写成单独的类，比如本例写成 (value,step) 元组。也可写成 (value,visited)，看自己喜好和题目；

已访问集合：为了避免队列中插入重复的值

BFS算法组成的套路：

初始化三元素：

Node = node(n) queue = [Node] visited = set([Node.value])

操作队列 —— 弹出队首节点：

vertex = queue.pop(0)

操作弹出的节点 —— 根据业务生成子节点（一个或多个）：

[node(vertex.value - n\*n, Node.step+1) for n in range(1,int(vertex.value\*\*.5)+1)]

判断这些节点 —— 符合业务条件，则return，不符合业务条件，且不在已访问集合，则追加到队尾，并加入已访问集合：

if i==0:

return new\_vertex.step

elif i not in visited:

queue.append(new\_vertex)

visited.add(i)```

若以上遍历完成仍未return，下面操作返回未找到代码：

return -1

[Reference]

[Revision history]