

# LeetCode 279

---

<https://leetcode.com/problems/perfect-squares/description/>

Yifeng Zeng

## Description

---

### 279. Perfect Squares

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

For example, given  $n = 12$ , return 3 because  $12 = 4 + 4 + 4$ ; given  $n = 13$ , return 2 because  $13 = 4 + 9$ .

## Idea Report

---

The primitive idea is just do a recursively search. For any number  $n_0$ , we can see if it can be split to another number  $n_1$  plus an perfect square number smaller than  $n_0$ . So this problem can be represent as for a given  $n_0$ , find  $x$  where  $x$  is smaller or equals  $n_0$  and is a perfect square. Then we can try to find the least number of perfect square numbers sum to  $n_1$ , plus 1 (our  $x$ ). We can treat  $n_1$  as  $n_0$  and recursively search the result, until  $n_0$  is a perfect square then we just return 1 and all the recursion levels will be returned. So we can just write a primitive DFS solution, and of course we need optimize it.

Code

```
class Solution {
    // Primitive DFS without memoization TLE
    public int numSquares(int n) {
        if (n <= 0) {
            return 0;
        }
        if (n == 1) {
            return 1;
        }
    }
}
```

```

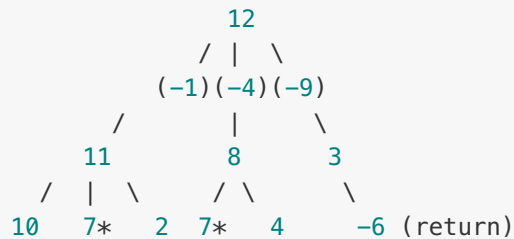
    }

    if ((int) Math.sqrt(n) * (int) Math.sqrt(n) == n) {
        return 1;
    }

    int min = Integer.MAX_VALUE;
    for (int i = 1; i * i < n; i++) {
        min = Math.min(min, numSquares(n - i * i) + 1);
        if (min == 1) {
            break;
        }
    }
    return min;
}
}

```

We can draw a solution space tree to see if there is any duplication.



We can see 7\* is a duplication, so we can have an array `int[] memo` to save the search result. `memo[i]` means the least number of perfect square numbers which sum to `i`.

Then we can add some memoization without much pruning.

```

class Solution {
    //DFS + memo + no much pruning TLE
    public int numSquares(int n) {
        int[] memo = new int[n + 1];
        return helper(n, memo);
    }

    private int helper(int n, int[] memo) {
        if (n <= 0) {
            return 0;
        }
        if ((int) Math.sqrt(n) * (int) Math.sqrt(n) == n) {

```

```

        return 1;
    }
    if (memo[n] != 0) {
        return memo[n];
    }
    int min = Integer.MAX_VALUE;
    for (int i = 1; i * i < n; i++) {
        min = Math.min(min, numSquares(n - i * i) + 1);
        if (min == 1) {
            break;
        }
    }
    return memo[n] = min;
}
}

```

Since we tried memo-search, we might want to try DP.

We can have an array `int[] f`, where `f[i]` means the least number of perfect square numbers which sum to `i`. So we want to find `f[n]`. We can initialize all the `f[perfect_square] = 1`. And for all other `f[i]`, it will be `f[i - j] + 1`, where `j` is a perfect square before `i`.

```

class Solution {
    // DP AC
    public int numSquares(int n) {
        int[] f = new int[n+1];
        for (int i = 1; i <= n; i++) {
            f[i] = i; // at most number i of 1s
            for (int j = 1; j * j <= i; j++) {
                f[i] = Math.min(f[i], f[i - j*j] + 1);
            }
        }
        return f[n];
    }
}

```

## Summary

---

- Hi TA, would you mind to give some ideas about how to do more pruning based on my second code, please? Thanks!