# 算法讨论班第二十五期——李玥珮

2016 年 3 月 4 日

## 63. Unique Paths II

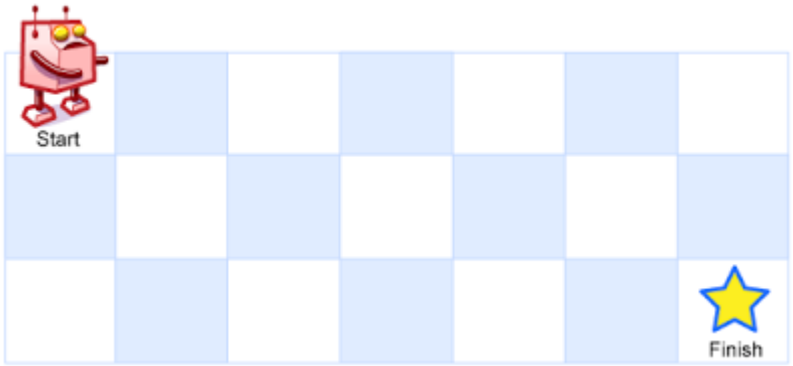
Follow up for "Unique Paths":
Now consider if some obstacles are added to the grids. How many unique paths would there be?
An obstacle and empty space is marked as 1 and 0 respectively in the grid.
For example,
There is one obstacle in the middle of a 3x3 grid as illustrated below.

```
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
```

The total number of unique paths is 2.

## 62. Unique Paths

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).
How many possible unique paths are there?




解题思路：

机器人只能向下与向右走，所以最后到达 finish 点的所有路径可以分为两种情况：经过 finish 左侧的方格向右一步到达终点，或者经过 finish 点上面的方格向下一步到达终点。将这个规律推广，我们可以得到下面的递推关系式：

$$s[i][j] = s[i-1][j] + s[i][j-1]$$

s[i][j] 是机器人从左上角出发到达 point[i][j] 的所有可能路径数。63 题只需要根据情况在有障碍的格子将路径数量设为 0（表示不能抵达）即可。DP 算法代码如下：

```
class Solution(object):
    def uniquePathsWithObstacles(self, obstacleGrid):
        """
        :type obstacleGrid: List[List[int]]
        :rtype: int
        """
        pathTable = copy.deepcopy(obstacleGrid)
        pathTable[0][0] = 1
        for i in range(len(obstacleGrid)):
            for j in range(len(obstacleGrid[0])):
                if obstacleGrid[i][j] == 1:
                    pathTable[i][j] = 0
                    continue
                if i == 0 and j == 0:
                    continue
                if i == 0:
                    pathTable[i][j] = pathTable[i][j-1]
                elif j == 0:
                    pathTable[i][j] = pathTable[i-1][j]
                else:
                    pathTable[i][j] = pathTable[i-1][j] + pathTable[i][j-1]

        return pathTable[len(obstacleGrid)-1][len(obstacleGrid[0])-1]
```

## 264. Ugly Number II

Write a program to find the `n`-th ugly number.

Ugly numbers are positive numbers whose prime factors only include `2, 3, 5`. For example, `1, 2, 3, 4, 5, 6, 8, 9, 10, 12` is the sequence of the first `10` ugly numbers.

Note that `1` is typically treated as an ugly number.

*Hint:*

1. *The naive approach is to call isUgly for every number until you reach the nth one. Most numbers are not ugly. Try to focus your effort on generating only the ugly ones.*
2. *An ugly number must be multiplied by either 2, 3, or 5 from a smaller ugly number.*
3. *The key is how to maintain the order of the ugly numbers. Try a similar approach of merging from three sorted lists: L1, L2, and L3.*
4. *Assume you have Uk, the kth ugly number. Then Uk+1 must be Min(L1 * 2, L2 * 3, L3 * 5).*

解题思路：

Hint 中的思路已经比较清晰，我们初始化一个序列用以存放 Ugly Number 的序列。初始时，序列中只含有一个数字 1（被强行规定为 Ugly 的数），之后每次用序列乘以 2、3、5 吗，在得到的序列中挑选最小值插入 UglyNumber 序列。算法代码如下：

```python
class Solution(object):
    def nthUglyNumber(self, n):
        """
        :type n: int
        :rtype: int
        """
        ugly = [1] * n
        i2 = i3 = i5 = -1
        x = v2 = v3 = v5 = 1
        for k in xrange(n):
            x = min(v2, v3, v5)
            ugly[k] = x
            if x == v2:
                i2 += 1
                v2 = ugly[i2] * 2
            if x == v3:
                i3 += 1
                v3 = ugly[i3] * 3
            if x == v5:
                i5 += 1
                v5 = ugly[i5] * 5
        return x
```

## 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`.

解题思路：

本题的思路与上一题类似，都是考虑动态规划的思路。假设我们已经知道了第 k 个数的 PerfectSquares 的个数 P[k]，求 P[k+1]。则可以得出递推公式：

$$P[k+1] = min\{P[i] + P[k+1-i]\} \quad 其中\ 1 <= i <= k$$

事实上，i 的取值并不需要由 1 连续的去到 k。我们只需要考虑 i 是平方数的情况就可以了。也就是：

$$i = v^2 \quad and\ i <= k$$

算法的代码如下:

```python
class Solution(object):
    numSquaresDP = [0]
    def numSquares(self, n):
        """
        :type n: int
        :rtype: int
        """
        if len(self.numSquaresDP) <= n:
            perfectSqr = [v**2 for v in xrange(1, int(math.sqrt(n)) + 1)]
            for i in xrange(len(self.numSquaresDP), n + 1):
                self.numSquaresDP.append( min(1 + self.numSquaresDP[i - sqr] for sqr
                    in perfectSqr if sqr <= i))
        return self.numSquaresDP[n]
```

理论背景：四平方和定理
每个正整数均可以表示为 4 个整数的平方和。