春节期间，读者留言最多的问题

Original labuladong labuladong 2021-02-23 16:20

读完本文，可以去力扣解决如下题目：

931.下降路径最小和（Medium）

这几天我抽空看了以前文章的留言，很多读者对动态规划问题的 base case、备忘录初始值等问题存在疑问。

本文就专门讲一讲这类问题，顺便聊一聊怎么通过题目的蛛丝马迹揣测出题人的小心思，辅助我们解题。

看下力扣第 931 题「下降路径最小和」，输入为一个 `n * n` 的二维数组 `matrix`，请你计算从第一行落到最后一行，经过的路径和最小为多少。

函数签名如下：

```
int minFallingPathSum(int[][] matrix);
```

就是说你可以站在 `matrix` 的第一行的任意一个元素，需要下降到最后一行。

每次下降，可以向下、向左下、向右下三个方向移动一格。也就是说，可以从 `matrix[i][j]` 降到 `matrix[i+1][j]` 或 `matrix[i+1][j-1]` 或 `matrix[i+1][j+1]` 三个位置。

请你计算下降的「最小路径和」，比如说题目给了一个例子：

示例 1：

```
输入：matrix = [[2,1,3],[6,5,4],[7,8,9]]
输出：13
解释：下面是两条和最小的下降路径，用加粗标注：
[[2,1,3],      [[2,1,3],
 [6,5,4],       [6,5,4],
 [7,8,9]]       [7,8,9]]
```

我们前文写过两道「路径和」相关的文章：动态规划之最小路径和 和 用动态规划算法通关魔塔。

今天这道题也是类似的，不算是困难的题目，所以**我们借这道题来讲讲 base case 的返回值、备忘录的初始值、索引越界情况的返回值如何确定**。

不过还是要通过 动态规划的标准套路 介绍一下这道题的解题思路，首先我们可以定义一个 `dp` 数组：

```
int dp(int[][] matrix, int i, int j);
```

这个 `dp` 函数的含义如下：

从第一行（`matrix[0][..]`）向下落，落到位置 `matrix[i][j]` 的最小路径和为 `dp(matrix, i, j)`。

根据这个定义，我们可以把主函数的逻辑写出来：

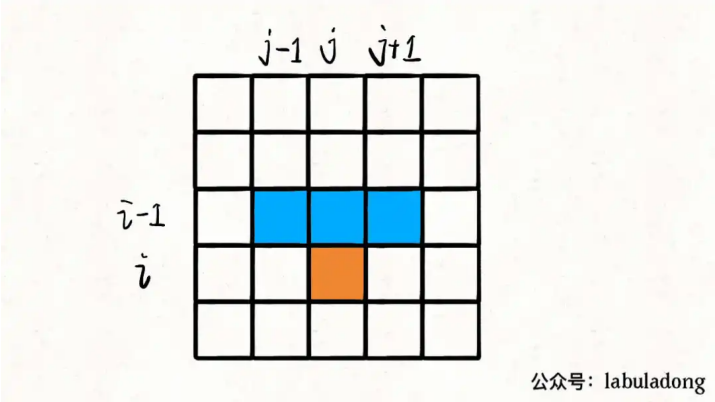
```
public int minFallingPathSum(int[][] matrix) {
    int n = matrix.length;
    int res = Integer.MAX_VALUE;

    // 终点可能在最后一行的任意一列
    for (int j = 0; j < n; j++) {
        res = Math.min(res, dp(matrix, n - 1, j));
    }

    return res;
}
```

因为我们可能落到最后一行的任意一列，所以要穷举一下，看看落到哪一列才能得

到最小的路径和。

接下来看看 dp 函数如何实现。

对于 matrix[i][j]，只有可能从 matrix[i-1][j], matrix[i-1][j-1], matrix[i-1][j+1] 这三个位置转移过来。



那么，只要知道到达 (i-1, j), (i-1, j-1), (i-1, j+1) 这三个位置的最小路径和，加上 matrix[i][j] 的值，就能够计算出来到达位置 (i, j) 的最小路径和：

```
int dp(int[][] matrix, int i, int j) {
    // 非法索引检查
    if (i < 0 || j < 0 ||
        i >= matrix.length ||
        j >= matrix[0].length) {
        // 返回一个特殊值
        return 99999;
    }
    // base case
    if (i == 0) {
        return matrix[i][j];
    }
    // 状态转移
    return matrix[i][j] + min(
            dp(matrix, i - 1, j),
            dp(matrix, i - 1, j - 1),
            dp(matrix, i - 1, j + 1)
        );
}

int min(int a, int b, int c) {
    return Math.min(a, Math.min(b, c));
```

```
}
```

当然，上述代码是暴力穷举解法，我们可以用备忘录的方法消除重叠子问题，完整代码如下：

```
public int minFallingPathSum(int[][] matrix) {
    int n = matrix.length;
    int res = Integer.MAX_VALUE;
    // 备忘录里的值初始化为 66666
    memo = new int[n][n];
    for (int i = 0; i < n; i++) {
        Arrays.fill(memo[i], 66666);
    }
    // 终点可能在 matrix[n-1] 的任意一列
    for (int j = 0; j < n; j++) {
        res = Math.min(res, dp(matrix, n - 1, j));
    }
    return res;
}

// 备忘录
int[][] memo;

int dp(int[][] matrix, int i, int j) {
    // 1、索引合法性检查
    if (i < 0 || j < 0 ||
        i >= matrix.length ||
        j >= matrix[0].length) {

        return 99999;
    }
    // 2、base case
    if (i == 0) {
        return matrix[0][j];
    }
    // 3、查找备忘录，防止重复计算
    if (memo[i][j] != 66666) {
        return memo[i][j];
    }
    // 进行状态转移
    memo[i][j] = matrix[i][j] + min(
            dp(matrix, i - 1, j),
            dp(matrix, i - 1, j - 1),
            dp(matrix, i - 1, j + 1)
        );

    return memo[i][j];
```

```
}

int min(int a, int b, int c) {
    return Math.min(a, Math.min(b, c));
}
```

如果看过我们公众号之前的动态规划系列文章，这个解题思路应该是非常容易理解的。

那么本文对于这个 dp 函数仔细探讨三个问题：

1、对于索引的合法性检测，返回值为什么是 99999？其他的值行不行？

2、base case 为什么是 `i == 0` ？

3、备忘录 `memo` 的初始值为什么是 66666？其他值行不行？

首先，说说 base case 为什么是 `i == 0` ，返回值为什么是 `matrix[0][j]` ，这是根据 dp 函数的定义所决定的。

回顾我们的 `dp` 函数定义：

从第一行（ `matrix[0][..]` ）向下落，落到位置 `matrix[i][j]` 的最小路径和为 `dp(matrix, i, j)` 。

根据这个定义，我们就是从 `matrix[0][j]` 开始下落。那如果我们想落到的目的地就是 `i == 0` ，所需的路径和当然就是 `matrix[0][j]` 呗。

再说说备忘录 memo 的初始值为什么是 66666，这是由题目给出的数据范围决定的。

备忘录 `memo` 数组的作用是什么？

就是防止重复计算，将 `dp(matrix, i, j)` 的计算结果存进 `memo[i][j]` ，遇到重复计算可以直接返回。

那么，我们必须要知道 `memo[i][j]` 到底存储计算结果没有，对吧？如果存结果

了，就直接返回；没存，就去递归计算。

所以，`memo` 的初始值一定得是特殊值，和合法的答案有所区分。

我们回过头看看题目给出的数据范围：

> `matrix` 是 `n * n` 的二维数组，其中 `1 <= n <= 100`；对于二维数组中的元素，有 `-100 <= matrix[i][j] <= 100`。

假设 `matrix` 的大小是 100 x 100，所有元素都是 100，那么从第一行往下落，得到的路径和就是 100 x 100 = 10000，也就是最大的合法答案。

类似的，依然假设 `matrix` 的大小是 100 x 100，所有元素是 -100，那么从第一行往下落，就得到了最小的合法答案 -100 x 100 = -10000。

也就是说，这个问题的合法结果会落在区间 `[-10000, 10000]` 中。

所以，我们 `memo` 的初始值就要避开区间 `[-10000, 10000]`，换句话说，`memo` 的初始值只要在区间 `(-inf, -10001] U [10001, +inf)` 中就可以。

最后，说说对于不合法的索引，返回值应该如何确定，这需要根据我们状态转移方程的逻辑确定。

对于这道题，状态转移的基本逻辑如下：

```
int dp(int[][] matrix, int i, int j) {

    return matrix[i][j] + min(
            dp(matrix, i - 1, j),
            dp(matrix, i - 1, j - 1),
            dp(matrix, i - 1, j + 1)
        );
}
```

显然，`i - 1, j - 1, j + 1` 这几个运算可能会造成索引越界，对于索引越界的 `dp` 函数，应该返回一个不可能被取到的值。

因为我们调用的是 `min` 函数，最终返回的值是最小值，所以对于不合法的索引，只要 `dp` 函数返回一个永远不会被取到的最大值即可。

刚才说了，合法答案的区间是 `[-10000, 10000]`，所以我们的返回值只要大于 10000 就相当于一个永不会取到的最大值。

换句话说，只要返回区间 `[10001, +inf)` 中的一个值，就能保证不会被取到。

至此，我们就把动态规划相关的三个细节问题举例说明了。

拓展延伸一下，建议大家做题时，除了题意本身，一定不要忽视题目给定的其他信息。

本文举的例子，测试用例数据范围可以确定「什么是特殊值」，从而帮助我们将思路转化成代码。

除此之外，数据范围还可以帮我们估算算法的时间/空间复杂度。

比如说，有的算法题，你只想到一个暴力求解思路，时间复杂度比较高。如果发现题目给定的数据量比较大，那么肯定可以说明这个求解思路有问题或者存在优化的空间。

除了数据范围，有时候题目还会限制我们算法的时间复杂度，这种信息其实也暗示着一些东西。

比如要求我们的算法复杂度是 `O(NlogN)`，你想想怎么才能搞出一个对数级别的复杂度呢？

肯定得用到 二分搜索 或者二叉树相关的数据结构，比如 `TreeMap`，`PriorityQueue` 之类的对吧。

再比如，有时候题目要求你的算法时间复杂度是 `O(MN)`，这可以联想到什么？

可以大胆猜测，常规解法是用 回溯算法 暴力穷举，但是更好的解法是动态规划，而且是一个二维动态规划，需要一个 `M * N` 的二维 `dp` 数组，所以产生了这样一个时间复杂度。

如果你早就胸有成竹了，那就当我没说，毕竟猜测也不一定准确；但如果你本来就没啥解题思路，那有了这些推测之后，最起码可以给你的思路一些方向吧？

总之，多动脑筋，不放过任何蛛丝马迹，你不成为刷题小能手才怪。

精华文章目录点这里 🔗

———————————————————

手把手刷动态规划 · 目录

上一篇

练琴时悟出的动态规划算法，帮我通关了《辐射4》

下一篇

学算法的大实话

Read more