动态规划

Orange

CQ No.11 High

2018 年 1 月 28 日

写在前面

虽然说是讲动态规划，但是可能会涉及到很多其它东西，甚至讲得比动态规划还深。所以你们看着办好啦。

大家一定要踊跃一点。

写在前面

虽然说是讲动态规划，但是可能会涉及到很多其它东西，甚至讲得比动态规划还深。所以你们看着办好啦。

大家一定要踊跃一点。

另外不要看《信息学奥赛一本通》，谢谢，不然你的动态规划只会学得一塌糊涂。

写在前面

虽然说是讲动态规划，但是可能会涉及到很多其它东西，甚至讲得比动态规划还深。所以你们看着办好啦。

大家一定要踊跃一点。

另外不要看《信息学奥赛一本通》，谢谢，不然你的动态规划只会学得一塌糊涂。

希望大家在最后能够感慨：啊动态规划就是这么个简单东西。

数字三角形问题

有一个由非负整数组成的三角形，形状与杨辉三角类似：第一行只有一个数，除了最下面的一行，每个数的左下方和右下方各有一个数。

从第一行的数开始，每次可以往左下或右下走一格，直到走到最下行。把沿途经过的数全部加起来，如何走才能使得这个和尽量大？

$$n \leq 1000$$

方法〇 —— 一个很好写的方法

对任意一个位置，哪一边的数字大，我们就走哪边。

方法〇 —— 一个很好写的方法

对任意一个位置，哪一边的数字大，我们就走哪边。
然而很明显这是错的。

方法〇 —— 一个很好写的方法

对任意一个位置，哪一边的数字大，我们就走哪边。

然而很明显这是错的。

我们把这种方法叫做 乱贪心。对于任何一个贪心的方法，在时间等条件允许时都要仔细证明。

方法

大家都会搜索吧？毫无疑问，搜索出来的答案一定是对的。但是时间复杂度如何呢？

方法

大家都会搜索吧？毫无疑问，搜索出来的答案一定是对的。但是时间复杂度如何呢？

总共有 2^{n-1} 条路径。

$$O(2^n)$$

方法

当然是动态规划啦！

从这个问题来看，我们使用动态规划的理由之一是解决某些爆搜难以解决的问题。

所以我们应当思考，为什么爆搜这么慢。

方法

我们来想想搜索是怎么搜的。

常见方法是记录从上往下的和，走到底时更新一下答案。但我们试试 反着思考，从下往上走。

假设我们正在第 i 行第 j 列。我们想的是，左边或者右边，哪边走出来的路径的和最大，我们就走哪边，或者是，从哪边来。因为我们在递归求解，所以我们每调用一次函数，就得到一次子问题的答案。

方法

我们来想想搜索是怎么搜的。

常见方法是记录从上往下的和，走到底时更新一下答案。但我们试试 反着思考，从下往上走。

假设我们正在第 i 行第 j 列。我们想的是，左边或者右边，哪边走出来的路径的和最大，我们就走哪边，或者是，从哪边来。因为我们在递归求解，所以我们每调用一次函数，就得到一次子问题的答案。

然后我们就得到了第 i 行第 j 列的答案，于是我们就高高兴兴地回到了第 $i-1$ 行。

方法

然后你就惊讶地发现：你会不止一次来到第 i 行第 j 列，因为从第 1 行第 1 列到它的路径不是唯一的。

然后你就又进行了许多次相同的运算……

方法

我们为什么不保存这个答案呢？

设 $f_{i,j}$ 表示第 i 行第 j 列的答案。如果我们已经运算出来了，我们就直接用保存的值。如果没有，那只好老老实实地来算了……

方法

我们为什么不保存这个答案呢？

设 $f_{i,j}$ 表示第 i 行第 j 列的答案。如果我们已经运算出来了，我们就直接用保存的值。如果没有，那只好老老实实地来算了……

看上去很诱人，但是效率如何呢？

方法

当一个位置已经计算出来后，我们不会再往下面走；而计算底层位置的答案只需要常数时间。

可以看作每次用最底层去更新倒数第二层，然后删去最底层。每次操作的时间复杂度为 $O(n)$，总共有 n 层，所以时间复杂度为 $O(n^2)$。

方法

以上方法的核心思想是：把已经算过的东西保存下来，不要重复计算；用已经算过的东西去计算没有算过的东西。

方法

以上方法的核心思想是：把已经算过的东西保存下来，不要重复计算；用已经算过的东西去计算没有算过的东西。

这就是动态规划的通俗描述。

方法

于是我们就有了这么个写法：在递归函数的开头检查是否已经算过。

方法

于是我们就有了这么个写法：在递归函数的开头检查是否已经算过。由于这种写法很像搜索，我们称这种写法为记忆化搜索，简称记搜。

方法

不难发现，我们可以直接从下往上推算，用两层循环就能搞定。
我们称这种写法为递推法。

方法

不难发现，我们可以直接从下往上推算，用两层循环就能搞定。

我们称这种写法为递推法。

递推时，我们从倒数第二层开始推算，用下一层的答案来更新当前层。这种用别人的答案来更新自己的答案的做法叫做填表法。有一个好记的名字，叫做“人人为我”。

方法

不难发现，我们可以直接从下往上推算，用两层循环就能搞定。

我们称这种写法为递推法。

递推时，我们从倒数第二层开始推算，用下一层的答案来更新当前层。这种用别人的答案来更新自己的答案的做法叫做填表法。有一个好记的名字，叫做“人人为我”。

不难发现，这种方法的时间复杂度和记忆化搜索一致。之前我们计算时间复杂度时其实就是用的这种方法。一般而言，递推法比记忆化搜索要快，因为它避免了递归的时间开销。

方法

不难发现，我们可以直接从下往上推算，用两层循环就能搞定。

我们称这种写法为递推法。

递推时，我们从倒数第二层开始推算，用下一层的答案来更新当前层。这种用别人的答案来更新自己的答案的做法叫做填表法。有一个好记的名字，叫做“人人为我”。

不难发现，这种方法的时间复杂度和记忆化搜索一致。之前我们计算时间复杂度时其实就是用的这种方法。一般而言，递推法比记忆化搜索要快，因为它避免了递归的时间开销。

但有时，有的问题难以用递推法解决；有的问题受题目性质影响，记忆化搜索不会遍历到所有状态，这种问题使用记搜反而快。

技巧与注意事项

memset 函数的使用？

技巧与注意事项

memset 函数的使用？
动态规划与“递推”的关系？

技巧与注意事项

memset 函数的使用？
动态规划与“递推”的关系？
是不是动态规划都像解决这个问题一样能够大幅降低时间复杂度？

技巧与注意事项

memset 函数的使用？
动态规划与“递推”的关系？
是不是动态规划都像解决这个问题一样能够大幅降低时间复杂度？
做很有可能是动态规划的题目的正确思考顺序？

动态规划的一些基本概念

对于任意一个非底层位置，

我们有两个选择：从左边来，或者从右边来，

选择答案最大的路径来更新当前位置的答案。

数字三角形问题 2

有一个由非负整数组成的三角形，形状与杨辉三角类似：第一行只有一个数，除了最下面的一行，每个数的左下方和右下方各有一个数。

从第一行的数开始，每次可以往左下或右下走一格，直到走到最下行。把沿途经过的数全部加起来，如何走才能使得这个和的个位 尽量大？

$$n \leq 1000$$

数字三角形问题 2

继续套用第一个问题的状态设计，发现会有一些问题。

数字三角形问题 2

继续套用第一个问题的状态设计，发现会有一些问题。
两个个位数最大的数加起来得到的数的个位不一定大。

数字三角形问题 2

继续套用第一个问题的状态设计，发现会有一些问题。
两个个位数最大的数加起来得到的数的个位不一定大。
我们称这种情况为不满足最优子结构性质。
动态规划的状态转移必须满足最优子结构性质。

数字三角形问题 2

设 $f_{i,j,k}$ 表示第 i 行第 j 列的数个位是否可能为 k。

数字三角形问题 2

设 $f_{i,j,k}$ 表示第 i 行第 j 列的数个位是否可能为 k。
转移显然，但是答案是什么？最底下的那一行的 f 又应是如何的？

数字三角形问题 2

设 $f_{i,j,k}$ 表示第 i 行第 j 列的数个位是否可能为 k。
转移显然，但是答案是什么？最底下的那一行的 f 又应是如何的？

这道题给我们的启发：动态规划必须做好三件事：

- 明确状态
- 知道边界的状态
- 能用状态算出我们需要的答案

数字三角形问题 2

这个算法的时间复杂度是多少？

数字三角形问题 2

这个算法的时间复杂度是多少？

$$O(100n^2)$$

数字三角形问题 2

这个算法的时间复杂度是多少？

$$O(100n^2)$$

一般来说，动态规划的时间复杂度为：

状态总数 × 决策个数 × 决策时间。

数字三角形问题 1 Ex

如果要求输出路径，该怎么做呢？

数字三角形问题 1 Ex

如果要求输出路径，该怎么做呢？

用 $pre_{i,j}$ 记录 $f_{i,j}$ 由哪个状态转移而来。因为我们的决策是选择左边或者选择右边，所以从哪儿转移而来就肯定选了哪个的。

数字三角形问题 1 Ex

如果要求输出路径，该怎么做呢？

用 $pre_{i,j}$ 记录 $f_{i,j}$ 由哪个状态转移而来。因为我们的决策是选择左边或者选择右边，所以从哪儿转移而来就肯定选了哪个的。

如果要求输出从下到上的路径，只能用栈保存路径。当然这也意味着你可以递归输出，因为递归的数据结构基础就是栈。

数字三角形问题 1 Ex

如果要求输出路径，该怎么做呢？

用 $pre_{i,j}$ 记录 $f_{i,j}$ 由哪个状态转移而来。因为我们的决策是选择左边或者选择右边，所以从哪儿转移而来就肯定选了哪个的。

如果要求输出从下到上的路径，只能用栈保存路径。当然这也意味着你可以递归输出，因为递归的数据结构基础就是栈。

绝大多数要输出一种具体方案的动态规划都是这么干的。

所以最基础的动态规划就这样啦！是不是很简单？
要进行接下来的学习，得先了解一点点图论的内容。

概念

图的逻辑概念为 $G = (V, E)$。
指的是图 G 由（顶）点集合 V 和边的集合 E 组成。

概念

图的逻辑概念为 $G = (V, E)$。
指的是图 G 由（顶）点集合 V 和边的集合 E 组成。

说大白话，让我给你们画一下就知道了。

分类

图大概可以分为两类：

- 有向图
- 无向图

分类

图大概可以分为两类：

- 有向图
- 无向图

其中有一种特殊的图，叫做有向无环图（Directed Acyclic Graph, DAG）。

无后效性

什么叫做无后效性呢？让我们回到数字三角形问题 1。

左边或者右边 选一条答案最大的。

如果某个状态的答案一旦计算出来就不会再改变，就称这个状态是无后效性的。

无后效性

什么叫做无后效性呢？让我们回到数字三角形问题 1。

左边或者右边 选一条答案最大的。

如果某个状态的答案一旦计算出来就不会再改变，就称这个状态是无后效性的。

DAG 是一种典型的无后效性的图，换句话说，从 **DAG** 上的任一点出发都不能回到出发点。

无后效性

什么叫做无后效性呢？让我们回到数字三角形问题 1。

左边或者右边 选一条答案最大的。

如果某个状态的答案一旦计算出来就不会再改变，就称这个状态是无后效性的。

DAG 是一种典型的无后效性的图，换句话说，从 **DAG** 上的任一点出发都不能回到出发点。

而 动态规划要求满足两个性质：最优子结构性质和无后效性，所以 DAG 和动态规划是息息相关的。

我太懒了，所以剩下的都没有做了……
凑合看看上课的笔记吧。