动态规划

陶鸿杰

QQ:1170755856

2019/05/05

数字三角形 (POJ1163)

在左侧的数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。

- 路径上的每一步都只能往左下或右下走。
- 只需要求出这个最大和即可，不必给出具体路径。
- 三角形的行数大于 1，小于等于 100，数字为 0 - 99。

问题分析:

- 三角形的行数大于 1，小于等于 100，数字为 0 - 99。
- 暴力的时间复杂度是 $O(2^n)$，而 2^{100} 肯定会超时
- 在不考虑记忆话搜索的情况下，我们需要怎么解决这个问题呢?

问题分析:

- 用 a[i][j] 存储这个数字三角形第 i 行的第 j 个数字
- 当前的位置 dp[i][j] 看成一个状态
- 则 dp[i][j] 的状态只能由它的上一层头顶上的那两个状态转移到的

问题分析:

- 我们已经知道对于任意一个位置，它的状态 dp[i][j] 是由它的上面两个状态转移过来的
- 我们反过来想，如果我们让路径从下往上走，那么：对于任意一个位置，它的状态 dp[i][j] 是由它的下面两个状态转移过来的

状态转移方程:

$$dp[i][j] = a[i][j] + max(dp[i+1][j], dp[i+1][j+1])$$

我们可以写出代码了

核心代码（复杂度为 $O(n^2)$）

```
for(int i=n-1;i>0;i--)
    for(int j=1;j<=i;j++)
^^Ia[i][j] += max(a[i+1][j],a[i+1][j+1]);
printf("%d\n",a[1][1]);
^^I
```

动态规划

动态规划解题的一般思路

1. 将原问题分解为子问题

- 把原问题分解为若干个子问题，子问题和原问题形式相同或类似，只不过规模变小了。子问题都解决，原问题即解决。
- 子问题的解一旦求出就会被保存，所以每个子问题只需求解一次。

动态规划解题的一般思路

2. 确定状态

- 动态规划解题时，我们往往将和子问题相关的各个变量的一组取值，称之为一个“状态”。
- 一个“状态”对应于一个或多个子问题，某个“状态”下的“值”，就是这个“状态”所对应的子问题的解。

动态规划解题的一般思路

3. 确定一些初始状态 (边界状态) 的值

- 就以“数字三角形”为例，初始状态就是底边数字，值就是底边数字值。

动态规划解题的一般思路

4. 确定状态转移方程

- 定义出什么是“状态”，以及在该“状态”下的“值”后，就要找出不同的状态之间如何迁移——即如何从一个或多个“值”已知的“状态”，求出另一个“状态”的“值”（“人人为我’递推型）。状态的迁移可以用递推公式表示，此递推公式也可被称作“状态转移方程”。

能用动规解决的问题的特点

问题具有最优子结构的性质

- 如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质

无后效性

- 当前的若干个状态值一旦确定，则此后过程的演变就只和这若干个状态的值有关，和之前是采取哪种手段或经过哪条路径演变到当前的这若干个状态，没有关系。

矩阵取数 (51Nod1083)

给定一个 m 行 n 列的矩阵，矩阵每个元素是一个正整数，你现在在左上角（第一行第一列），你需要走到右下角（第 m 行，第 n 列），每次只能朝右或者下走到相邻的位置，不能走出矩阵。走过的数的总和作为你的得分，求最大的得分。

问题分析

贪心可以么?
我们先把大的数字走完!
但是....
无论我们以什么方式走到 3，总和都是 $1+1+3+1+1+1+1=9$

问题分析

我们为了 1 个 3，放弃了那么多个 2，不值啊
如果我们放弃 3 而走那些 2，得到的和是 $1+1+2+2+2+1+1=10$

看来贪心是错的！因为我们走到最大值时有很多值不能走到了，一个最大值会把我们带沟里去

问题分析

你觉得枚举可行么?
你需要往下走 (m −1) 次，往右走 (n −1) 次，才能走到右下角，一共走 m + n −2 步！可行路径的总条数有 C_{m+n-2}^{m-1}。

问题分析

当矩阵的行数和列数都等于 100 的时候，来让我们计算一下:

$C_{1198}^{99} =$
$22750883079422934966181954039568885395604168260154104734000$

dp 一下

- 用 a[i][j] 存储这个二维矩阵的第 i 行第 j 个的数字
- 当前的位置 dp[i][j] 看成一个状态
- 则 dp[i][j] 的状态只能由它的头顶上的状态和它左侧的那个状态转移到的

dp 一下

那么我们就可以写出来状态转移方程了
状态转移方程如下所示：

状态转移方程

核心代码

代码实现（复杂度为 $O(n^2)$）

```
for(int i=1;i<=n;i++)
  for(int j=1;j<=n;j++)
    if(i==1)
^^Ia[i][j] += a[i][j-1];
    else if(j==1)
^^Ia[i][j] += a[i-1][j];
    else
^^Ia[i][j] += max(a[i-1][j],a[i][j-1]);
printf("%d\n",a[n][n]);
```

状态分析

代码分析 (i=1 时)

对于第 j 列，它的状态只能由它左侧的状态来传递的
即 $dp[i][j] = a[i][j] + dp[i][j-1]$
箭头的方向即为状态转移的方向

状态分析

代码分析 (i=2 时)

对于第 j 列，它的状态由它左侧的状态和它上面的状态来传递的

即 $dp[i][j] = a[i][j] + max(dp[i][j-1], dp[i-1][j])$

箭头的方向即为状态转移的方向

状态分析

代码分析 (i=3 时)

对于第 j 列，它的状态由它左侧的状态和它上面的状态来传递的

即 $dp[i][j] = a[i][j] + max(dp[i][j-1], dp[i-1][j])$

箭头的方向即为状态转移的方向

最大子段和 (51Nod1049)

题目描述

给出一个整数数组 a(正负数都有)，如何找出一个连续子数组(可以一个都不取，那么结果为 0)，使得其中的和最大?
例如：-2,11,-4,13,-5,-2，和最大的子段为：11,-4,13。和为 20。

输入

第 1 行：整数序列的长度 N（$2 <= N <= 50000$）
第 2 - N + 1 行：N 个整数（$-10^9 <= A[i] <= 10^9$）

输出

输出最大子段和。

问题分析

例如

-2,11,-4,13,-5,-2
和最大的子段为：11,-4,13。和为 20。

枚举?
让数组 a_i 保存第 i 个数字，那么，我们枚举的话，会得到这样的结果:

- $\{a_1\}, \{a_1 + a_2\}, \{a_1 + a_2 + a_3\}, \{a_1 + a_2 + a_3 + a_4\}, \{a_1 + a_2 + a_3 + a_4 + a_5\}$
- $\{a_2\}, \{a_2 + a_3\}, \{a_2 + a_3 + a_4\}, \{a_2 + a_3 + a_4 + a_5\}$
- $\{a_3\}, \{a_3 + a_4\}, \{a_3 + a_4 + a_5\}$
- $\{a_4\}, \{a_4 + a_5\}$
- $\{a_5\}$

问题分析

例如

-2,11,-4,13,-5,-2
和最大的子段为：11,-4,13。和为 20。

- $\{a_1\}, \{a_1 + a_2\}, \{a_1 + a_2 + a_3\}, \{a_1 + a_2 + a_3 + a_4\}, \{a_1 + a_2 + a_3 + a_4 + a_5\}$
- $\{a_2\}, \{a_2 + a_3\}, \{a_2 + a_3 + a_4\}, \{a_2 + a_3 + a_4 + a_5\}$
- $\{a_3\}, \{a_3 + a_4\}, \{a_3 + a_4 + a_5\}$
- $\{a_4\}, \{a_4 + a_5\}$
- $\{a_5\}$

我们需要枚举起点，终点，以及计算它们的和，时间复杂度是 $O(n^3)$
即使我们利用前缀和的方法，也只能把时间复杂度降低到 $O(n^2)$，还是比较慢

问题分析

例如

-2,11,-4,13,-5,-2
和最大的子段为：11,-4,13。和为 20。

- $\{a_1\}, \{a_1 + a_2\}, \{a_1 + a_2 + a_3\}, \{a_1 + a_2 + a_3 + a_4\}, \{a_1 + a_2 + a_3 + a_4 + a_5\}$
- $\{a_2\}, \{a_2 + a_3\}, \{a_2 + a_3 + a_4\}, \{a_2 + a_3 + a_4 + a_5\}$
- $\{a_3\}, \{a_3 + a_4\}, \{a_3 + a_4 + a_5\}$
- $\{a_4\}, \{a_4 + a_5\}$
- $\{a_5\}$

能不能在 $O(n)$ 的时间内把这个问题解决?

问题分析

例如

-2,11,-4,13,-5,-2
和最大的子段为：11,-4,13。和为 20。

- $\{a_1\}, \{a_1 + a_2\}, \{a_1 + a_2 + a_3\}, \{a_1 + a_2 + a_3 + a_4\}, \{a_1 + a_2 + a_3 + a_4 + a_5\}$
- $\{a_2\}, \{a_2 + a_3\}, \{a_2 + a_3 + a_4\}, \{a_2 + a_3 + a_4 + a_5\}$
- $\{a_3\}, \{a_3 + a_4\}, \{a_3 + a_4 + a_5\}$
- $\{a_4\}, \{a_4 + a_5\}$
- $\{a_5\}$

用动态规划的思路想一下:
如果我们用 dp[i] 记录第 i 个位置上的状态
让 dp[i] 表示以 a[i] 为结尾的最大子段和

问题分析

例如

-2,11,-4,13,-5,-2
和最大的子段为：11,-4,13。和为 20。

- $\{a_1\}, \{a_1+a_2\}, \{a_1+a_2+a_3\}, \{a_1+a_2+a_3+a_4\}, \{a_1+a_2+a_3+a_4+a_5\}$
- $\{a_2\}, \{a_2+a_3\}, \{a_2+a_3+a_4\}, \{a_2+a_3+a_4+a_5\}$
- $\{a_3\}, \{a_3+a_4\}, \{a_3+a_4+a_5\}$
- $\{a_4\}, \{a_4+a_5\}$
- $\{a_5\}$

那么对于第 i 个数字 a[i], 我们可以取, 也可以不取
如果加上 a[i] 后, $dp[i] < 0$, 那么就不能取 a[i], 这个时候 dp[i]=0
否则 dp[i] 就等于 dp[i-1]+a[i], 这个就是以 a[i] 为结尾的最大子段和

问题分析

例如

-2,11,-4,13,-5,-2
和最大的子段为：11,-4,13。和为 20。

- dp 数组初始化为 0
- dp[i] = max(dp[i-1]+a[i], dp[i])
- 遍历一遍 dp 数组，取最大的

DP 代码

代码实现（复杂度为 $O(n)$）

```
memset(dp,0,sizeof(dp));
ll maxx = 0;
for(int i=1;i<=n;i++)
{
    cin>>a[i];
    dp[i] = max(dp[i-1]+a[i],a[i]);
    maxx = max(maxx,dp[i]);
}
cout<<maxx<<endl;
^^I
```

最长公共子序列 (POJ 1458)

最长公共子序列定义

最长公共子串（Longest Common Substring）和最长公共子序列（Longest Common Subsequence, LCS）的区别：
子串（Substring）是串的一个连续的部分，子序列（Subsequence）则是从不改变序列的顺序，而从序列中去掉任意的元素而获得的新序列；
更简略地说，前者（子串）的字符的位置必须连续，后者（子序列 LCS）则不必。
比如字符串 acdfg 同 akdfc 的最长公共子串为 df，而他们的最长公共子序列是 adf。LCS 可以使用动态规划法解决。

最长公共子序列 (POJ 1458)

题目描述

给出两个字符串 A B，求 A 与 B 的最长公共子序列的长度。
比如两个串为：

- abcfbc
- abfcab

ab 是两个串的子序列，abc 也是，abfc 也是，其中 abfc 是这两个字符串最长的子序列。
所以结果是 4

最长公共子序列 (POJ 1458)

DP 一下

不妨定义 $dp[i][j]$ 表示 $A_1 \cdots A_i$ 和 $B_1 \cdots B_j$ 对应的 LCS 的长度
那么, $A_1 \cdots A_{i+1}$ 和 $B_1 \cdots B_{j+1}$ 对应的公共子序列可能是

- 当 $A_{i+1} = B_{j+1}$ 时, 在 $A_1 \cdots A_i$ 和 $B_1 \cdots B_j$ 的公共子序列末尾追加上 A_{i+1}
- $A_1 \cdots A_i$ 和 $B_1 \cdots B_{j+1}$ 的公共子序列
- $A_1 \cdots A_{i+1}$ 和 $B_1 \cdots B_j$ 的公共子序列

三者中的某一个，所以就有了一下递推关系成立

$$dp[i+1][j+1] = \begin{cases} dp[i][j]+1 & A_{i+1} = B_{j+1} \\ max(dp[i+1][j], dp[i][j+1]) & else \end{cases}$$

DP 代码

代码实现（复杂度为 $O(n*m)$）

```
for(int i=0;i<n;++i)
  for(int j=0;j<m;++j)
    if(a[i]==b[j])
      dp[i+1][j+1] = dp[i][j] + 1;
    else
      dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1]);
cout<<dp[n][m]<<endl;
^^I
```

最长上升子序列

最长上升子序列问题

有一个长为 n 的数列 $a_0, a_1, \ldots, a_{n-1}$。请求出这个序列中最长的上升子序列的长度。上升子序列指的是对于任意的 $i < j$ 都满足 $a_i < a_j$ 的子序列。

- $1 \leq n \leq 1000$
- $0 \leq a_i \leq 1000000$

输入

5

4 2 3 1 5

输出

3 (a_1, a_2, a_4 构成的子序列 $2, 3, 5$ 最长)

最长上升子序列

这一问题能够通过使用 DP 在 $O(n^2)$ 的时间内解决，我们来建立一下递推关系。

- 定义 dp[i]：以 a_i 为末尾的最长上升子序列的长度

以 a_i 为结尾的上升子序列是

- 只包含 a_i 的子序列
- 在满足 $j < i$ 并且 $a_j < a_i$ 的以 a_j 为结尾的上升子序列末尾，追加上 a_i 后得到的子序列

这二者之一。这样我们就到到了以下的递推关系：

$$dp[i] = max\{1, dp[j] + 1 | j < i且a_j < a_i\}$$

代码实现

DP 代码

```
int res = 0;
for(int i=0;i<n;++i)
{
  dp[i] = 1;
  for(int j=0;j<i;++j)
    if(a[j]<a[i])
      dp[i] = max(dp[i], dp[j]+1);
  res = max(res, dp[i]);
}
cout<<res<<endl;
^^I
```

结束

学习算法需要多练多思考，这里给大家推荐几个学习动态规划地方 (点击即可转到)：

- PKU 郭炜老师讲的算法课程
- 51Nod 里面的教程
- 夜深人静写算法（二）- 动态规划