动态规划

laofu

陈江伦 (清华大学交叉信息研究院)

July 18, 2021

单调队列一般运用于区间 DP 的转移优化，它一般针对的状态具有两个状态，两个状态之间没有优劣关系。
例题：求序列 a 中所有长度为 d 的子区间的最大值。
我们从前往后扫描，对于一个元素它有两个特征: 位置和大小，一个数越大，它越有可能成为最大值，一个数的位置越靠后，它会在之后利用的概率更高，所以我们维护一个从前往后元素递减位置递增的队列，每次当发现队列首端超过距离范围时就将其弹出。

斜率优化

斜率优化是决策单调性动态规划的一个特殊形式。这类问题的 DP 转移式一般都可以化为一次函数的形式，并且它的斜率是单调的，那么我们通过维护凸包来求得 DP 值在什么时候取得最优解。

Problem

例题：求解

$f_i = \min_{j<i}(g_j + (\sum_{k=j+1}^{i} c_k)^2)$

Problem

例题：求解

$f_i = \min_{j<i}(g_j + (\sum_{k=j+1}^{i} c_k)^2)$

$f_i = \min_{j<i} g_j + (sum_i - sum_j)^2$

$g_j + sum_j^2 = 2sum_i sum_j + f_i - sum_i^2$

把 $(sum_j, g_j + sum_j^2)$ 看出二维平面上一个点，有一条斜率为 $2sum_i$ 的直线，要最小化直线的截距。

Problem

例题：求解
$f_i = \min_{j<i}(g_j + (\sum_{k=j+1}^{i} c_k)^2)$
$f_i = \min_{j<i} g_j + (sum_i - sum_j)^2$
$g_j + sum_j^2 = 2sum_i sum_j + f_i - sum_i^2$
把 $(sum_j, g_j + sum_j^2)$ 看出二维平面上一个点，有一条斜率为 $2sum_i$ 的直线，要最小化直线的截距。
很显然，若干要让截距尽可能小，那么点一定位于点集的下凸壳上。那么我们可以在凸壳上三分找到最优解的位置。
不过我们还需要利用这个问题的特殊性质：sum_j 是按照标号单调递增的，所以我们可以动态加点维护凸包，同时 $2sum_i$ 也是按照标号递增的，也就是说直线的斜率递增。
画图可以发现，当直线增加时，在下凸壳上取得截距最小的位置也是单调不降的。所以我们可以用一个单调队列维护凸包。

01 背包问题

有若干物品，每个物品有一个体积和价值，你有一个指定容量的背包，求能够装的物品的最大价值。

01 背包问题

有若干物品，每个物品有一个体积和价值，你有一个指定容量的背包，求能够装的物品的最大价值。

$f[i][j]$ 表示只考虑前 i 个物品，用 j 单位容量能装物品的最大价值。

Algorithm 2 01 背包问题

$f[0\cdots n][0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i][j] \leftarrow \max(f[i-1][j], f[i-1][j-v_i]+w_i)$
  end for
end for

空间优化

Algorithm 3 滚动数组

$f[0 \cdots 1][0 \cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i\&1][j] \leftarrow \max(f[1-(i\&1)][j], f[1-(i\&1)][j-v_i]+w_i)$
  end for
end for

Algorithm 4 滑动数组

$f[0 \cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from m downto v_i **do**
    $f[j] \leftarrow \max(f[j], f[j-v_i]+w_i)$
  end for
end for

完全背包问题

有若干类物品，每类物品有一个体积和价值，数量有无限个，你有一个指定容量的背包，求能够装的物品的最大价值。

完全背包问题

有若干类物品，每类物品有一个体积和价值，数量有无限个，你有一个指定容量的背包，求能够装的物品的最大价值。
$f[i][j]$ 表示只考虑前 i 个物品，用 j 单位容量能装物品的最大价值。

Algorithm 6 01 背包问题

$f[0\cdots n][0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i][j] \leftarrow \max(f[i-1][j], f[i][j-v_i]+w_i)$
  end for
end for

空间优化

Algorithm 7 滚动数组

$f[0\cdots 1][0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[i\&1][j] \leftarrow \max(f[1-(i\&1)][j], f[i\&1][j-v_i]+w_i)$
  end for
end for

Algorithm 8 滑动数组

$f[0\cdots m] \leftarrow 0$
for i from 1 to n **do**
  for j from v_i to m **do**
    $f[j] \leftarrow \max(f[j], f[j-v_i]+w_i)$
  end for
end for

多重背包问题

有若干类物品，每类物品有一个体积和价值，每类物品有一定数量，你有一个指定容量的背包，求能够装的物品的最大价值。

算法一

把一个可以使用 k 次的物品看成若干个体积和价值都为这个物品的 2 的整数次幂倍的 01 物品。
显然根据二进制原理这个拆分数不超过 log 个。那么对这 log 个物品做 01 背包即可。

单调队列

$$f[i][j] = \max_{t \in [0,k]} (f[i-1][j-t*v]+t*w)$$

把模 v 意义下相同的 j 放在一起做，那么问题就变成了求所有长度为 k 的区间的最大值，单调队列就可以维护。

依赖背包问题

有若干个物品形成一棵树，每个物品有一个体积和价值，你有一个指定容量的背包，要求如果选了某个物品，则它的父亲也必须被选择，求能装的物品的最大价值。

依赖背包问题

有若干个物品形成一棵树，每个物品有一个体积和价值，你有一个指定容量的背包，要求如果选了某个物品，则它的父亲也必须被选择，求能装的物品的最大价值。
首先我们把树的 DFS 序造出来。
转移时，如果选择了某个物品，则可以正常向后转移，如果未选择某个物品，则它的子树中不能选择任何物品，而 dfs 序上一棵子树是一段连续的区间，我们直接跳过这个区间转移到之后的状态即可。

树形背包复杂度

把背包问题放到树上，$f[i][j]$ 表示在 i 的子树中的最大收益，每次的转移就是合并两棵子树。
假设节点 i 的每个孩子的子树大小为 $s_1 \cdots s_n$，则转移复杂度为

$$
\begin{aligned}
\sum_{i=2}^{n} s_i \sum_{j=1}^{i-1} s_j &= \sum_{i=1}^{n} \left(s_i \times \sum_{j=1}^{n} s_j - s_i^2 \right) \\
&\le \sum_{i=1}^{n} \left(s_i \sum_{j=1}^{n} s_i - s_i^2 \right) \\
&= \sum_{i=1}^{n} \left(s_i \times sum - s_i^2 \right) \\
&= sum^2 - \sum_{i=1}^{n} s_i^2
\end{aligned}
$$

数位 DP

数位 DP 是一类计数问题的统称，它要求记录在一个大小范围内的数/字符串的某种权值。
比如最常见的数位 DP 是：求在一段区间 [l,r] 内满足一定条件的数的个数。
首先，面对这种问题一般都会先把两个限制拆成一个，求 [0,r] 内满足条件的个数减去 [0,l-1] 内满足条件的个数。
对于这种只有上限的数位 DP，有两种基本的方法。

方法一

我们把上限 n 先数位分解为 $n[0..m]$，从高往低位扫，扫到第 k 位时我们统计以 $n[k+1\cdots m]$ 为前缀，且第 k 位严格小于 n[k] 的数的信息和。如果这项信息能够 O(1) 直接计算就可以带入，如果不能，可以先预处理一个数组 f[i][j] 表示最高位 a[i]=j，且 $0\sim i-1$ 位都任意填的所有数的信息和。

方法二

方法 1 有一定的局限性，它要求对于 $n[k+1..m]$ 这些数位的信息要能够和 $0..k-1$ 位的信息快速合并。还有另一种记忆化搜索的方式可以有效解决这个问题。我们从高位往低位搜索，每次枚举某一位填的数，然后用 $pre[i][S]$ 表示 $0 \sim i$ 位任意填，同时高位的数的信息状态为 S 的信息总和。在 dfs 时我们还加一个 bool 参数 t，表示当前枚举的高位是否严格等于 n。如果 t 为 0，那么可以对这一部分记忆化。如果 t 为 1，那么可以递归下去。我们注意到最多只有 o(位数) 个状态的 t=1，所以这一部分做一次的复杂度是 o(位数) 的，其它部分进行记忆化，复杂度是 O(状态数) 的

Problem

我们称一个正整数 n 是好的，当且仅当 n 能被他的所有数位整除。现在需要计算在一个给定的区间 [l, r] 中的好数的个数