线性动态规划

李淳风

长郡中学

2024 年 7 月 29 日

前言

我们之前探讨过很多算法，都是利用问题的可划分性以及子问题之间的相似性来进行归纳，降低求解的复杂度，动态规划也不例外。动态规划把原问题视作若干个重叠子问题的逐层推进，每个子问题的求解过程都构成一个“阶段”。在完成前一个阶段的计算后，动态规划才会执行下一阶段的计算。

为了保证这些计算能够按照顺序、不重复地进行，动态规划要求已经求解的子问题不受后续阶段的影响，也被称为“无后效性”。换言之，如果我们把问题中的“状态”看作节点，“状态的转移”看成边，那么就会构成一张有向无环图，动态规划的求解过程就是该有向无环图的一个拓扑序。

在很多情况下，动态规划用于求解最优化问题。此时，下一阶段的最优解应该能够由前面各阶段子问题的最优解推出。这个条件就是“最优子结构”性质。实际上这告诉我们，动态规划只对每个状态保留了与解集相关的部分代表信息，我们要能够通过这些代表信息不断去导出后续阶段的代表信息，动态规划才能起到优化作用。

“状态”“阶段”“决策”是构成动态规划算法的三要素，而“子问题重叠性”“无后效性”“最优子结构性质”是问题能用动态规划求解的三个条件。

介绍

我们在这里把具有线性“阶段”划分的动态规划算法统称为线性 DP。
相信大家对最长上升子序列（LIS）、最长公共子序列（LCS）、数字三角形等 DP 的经典入门例题已经不再陌生了。
容易发现，状态无论是一维还是多维，DP 算法在这些问题上都体现为“作用在线性空间上的递推”——DP 的阶段沿着各个维度线性增长。

例题

Mr.Young's Picture Permutatioins

有 n 个学生合影，左端对齐站成了 k 排，每排有 $n_1, n_2, \cdots, n_k$ 个人。第 1 排在最后，第 k 排在最前。学生的身高互不相同，把他们从高到低依次标记为 $1, 2, \cdots, n$。合影时要求每一排从左到右身高递减，每一列从后到前身高递减。

现在确定了 k 和 $n_1, n_2, \cdots, n_k$，求有多少种合影位置。$n \le 30, k \le 5$。

例题

Mr.Young's Picture Permutatioins

有 n 个学生合影，左端对齐站成了 k 排，每排有 $n_1, n_2, \cdots, n_k$ 个人。第 1 排在最后，第 k 排在最前。学生的身高互不相同，把他们从高到低依次标记为 $1, 2, \cdots, n$。合影时要求每一排从左到右身高递减，每一列从后到前身高递减。
现在确定了 k 和 $n_1, n_2, \cdots, n_k$，求有多少种合影位置。$n \le 30, k \le 5$。

因为在合法的方案中，每行每列的身高都是单调的，故我们可以从高到低依次考虑标记为 $1, 2, \cdots, n$ 的学生所处的位置。这样一来，在任意时刻，已经安排好位置的学生，在每一行中所处的一定是从左端开始的连续若干个位置。所以，我们可以用一个 k 元组 $(a_1, a_2, \cdots, a_k)$ 表示每一行已经安排的学生人数。

Mr.Young's Picture Permutatioins

当安排一名新的学生时，如果他可以站到第 i 行，则需要满足 $i=1$ 或者 $a_i > a_{i-1}$。这样就能保证每一列的单调性，至于行的单调性，通过学生的枚举顺序就能保证。
所以，我们并不关心已经站好的 $a_1+a_2+\cdots+a_k$ 名学生具体是怎么站的，k 元组 $(a_1, a_2, \cdots, a_k)$ 足以表示当下的状态，描述一个子问题。
因此，我们可以把 $a_1, a_2, \cdots, a_k$ 作为阶段，每次安排一名新的学生时，$a_1, a_2, \cdots, a_k$ 会加一，从而转移到后续的阶段，各个维度线性增长。
具体地，我们用 $f[a_1, a_2, a_3, a_4, a_5]$ 表示各排从左端起分别站了 a_1, a_2, a_3, a_4, a_5 个人的合影方案数。$k<5$ 的情况可以通过添加人数上限为 0 的排来补足。
边界为 $f[0,0,0,0,0]=1$，其余均为 0。
若 $a_1 < n_1$，则 $f[a_1+1, a_2, a_3, a_4, a_5] += f[a_1, a_2, a_3, a_4, a_5]$。
若 $a_2 < n_2$ 且 $a_2 < a_1$，则 $f[a_1, a_2+1, a_3, a_4, a_5] += f[a_1, a_2, a_3, a_4, a_5]$。
其余排同理。
表示完状态之后，转移方程有两种形式：当下状态从哪些状态得来，以及当下状态可以走到哪些状态。两种方法并无太大区别，一般是哪种方便用哪种。

例题

LCIS 最长公共上升子序列

对于两个序列 a 和 b，如果它们都包含一段位置不一定连续的数，且数值是严格递增的，那么称这一段数是两个数列的公共上升子序列。所有的公共上升子序列中最长的，就是最长公共上升子序列。
给定 a 和 b，请你求出它们的最长公共上升子序列的长度。
数列 a 和 b 的长度均不超过 3000。

例题

LCIS 最长公共上升子序列

对于两个序列 a 和 b，如果它们都包含一段位置不一定连续的数，且数值是严格递增的，那么称这一段数是两个数列的公共上升子序列。所有的公共上升子序列中最长的，就是最长公共上升子序列。
给定 a 和 b，请你求出它们的最长公共上升子序列的长度。
数列 a 和 b 的长度均不超过 3000。

这道题是 LIS 和 LCS 的综合。回想之前这两个问题的做法，我们很容易想到下列解法：
$f[i,j]$ 表示 $a_1 \sim a_i$ 与 $b_1 \sim b_j$ 可以构成的以 b_j 为结尾的 LCIS 的长度。
当 $a_i \neq b_j$ 时，有 $f[i,j] = f[i-1,j]$；
当 $a_i = b_j$ 时，有（假设 $a_0 = b_0 = -\infty$）：

$$f[i,j] = \max_{0 \le k < j, b_k < b_j} \{f[i-1,k]\} + 1 = \max_{0 \le k < j, b_k < a_i} \{f[i-1,k]\} + 1$$

LCIS 最长公共上升子序列

显然，上述做法可以使用三重循环来计算：

```
for(int i=1;i<=n;i++)
  for(int j=1;j<=m;j++)
    if(a[i]==b[j]){
      for(int k=0;k<j;k++)
          if(b[k]<a[i])
            f[i][j]=max(f[i][j],f[i-1][k]+1);
    }
    else f[i][j]=f[i-1][j];
```

LCIS 最长公共上升子序列

显然，上述做法可以使用三重循环来计算：

```
for(int i=1;i<=n;i++)
  for(int j=1;j<=m;j++)
    if(a[i]==b[j]){
      for(int k=0;k<j;k++)
          if(b[k]<a[i])
            f[i][j]=max(f[i][j],f[i-1][k]+1);
    }
    else f[i][j]=f[i-1][j];
```

在转移过程中，我们把满足 $0 \le k < j, b_k < a_i$ 的 k 构成的集合称为 $f[i,j]$ 进行状态转移时的决策集合，记为 $S(i,j)$。注意到，在第二层循环 j 从 1 增加到 m 时，第一层循环 i 是一个定值，这使得 $b_k < a_i$ 这个条件是固定的。也就一位置，当 j 变为 $j+1$ 时，原本符合条件的 k 仍然符合条件，此时只有 j 有可能新进入决策集合。

因此，我们每次只需要 $O(1)$ 地检查条件 $b_j < a_i$ 是否满足，即可快速维护决策集合 $S(i,j)$ 中 $f[i-1,k]$ 地最大值。复杂度变为 $O(n^2)$。

例题

Making the Grade

给定长度为 n 的序列 a，构造一个长度为 n 的序列 b，满足：

- b 非严格单调，即 $b_1 \le b_2 \le \cdots \le b_n$ 或 $b_1 \ge b_2 \ge \cdots \ge b_n$。
- 最小化 $S = \sum_{i=1}^{n} |a_i - b_i|$。

现在要求出这个最小值 S。$1 \le n \le 2000, 1 \le |a_i| \le 10^9$。

例题

Making the Grade

给定长度为 n 的序列 a，构造一个长度为 n 的序列 b，满足：

- b 非严格单调，即 $b_1 \le b_2 \le \cdots \le b_n$ 或 $b_1 \ge b_2 \ge \cdots \ge b_n$。
- 最小化 $S = \sum_{i=1}^{n} |a_i - b_i|$。

现在要求出这个最小值 S。$1 \le n \le 2000, 1 \le |a_i| \le 10^9$。

序列 b 有“非严格递增”和“非严格递减”两种情况，我们分别进行一次求解。下面以递增为例。

首先，在满足 S 最小的情况下，一定存在一种构造序列 b 的方案，使得 b 中的数值都在 a 中出现过。

可以使用数学归纳法证明。$n = 1$ 时显然成立。现在假设对 $n = k - 1$ 成立，此时构造出的序列为 $b_1 \sim b_{k-1}$。

当 $n = k$ 时，如果 $b_{k-1} \le a_k$，则令 $b_k = a_k$ 即可。否则，要么 $b_k = b_{k-1}$，要么存在一个 j 使得 $b_j, b_{j+1}, \cdots, b_k$ 为同一个值 v。设 $a_j, a_{j+1}, \cdots, a_k$ 的中位数是 mid，若 $mid \ge b_{j-1}$，则 $v = mid$，否则 $v = b_{j-1}$。但不管哪种情况，$b_1 \sim b_k$ 中的所有数都在 a 中出现过。

Making the Grade

有了之前的结论，我们就方便设计动态规划了。设 f_i 表示完成了前 i 个数的构造，并且 $b_i = a_i$ 时，S 的最小值。

$$f_i = \min_{0 \le j < i, a_j \le a_i} \{f_j + cost(j+1, i-1)\}$$

其中 $cost(j+1, i-1)$ 表示在满足 $a_j \le b_{j+1} \le \cdots \le b_{i-1} \le a_i$ 的条件下，构造 $b_{j+1}, \cdots, b_{i-1}$ 且，$\sum_{k=j_1}^{i-1} |a_k - b_k|$ 的最小值。
而在计算 $cost(j+1, i-1)$ 的时候，我们只需要计算让 $b_{j+1}, \cdots, b_{i-1}$ 中前一部分变为 a_j，后一部分变为 a_i 的最小代价即可。因为如果在 $b_{j+1}, \cdots, b_{i-1}$ 中有 $a_k (j < k < i)$，那么在 $j = k$ 时已经被考虑过了。
采用朴素方法计算 $cost$ 值的话，复杂度为 $O(n^3)$。

Making the Grade

既然只把“已经处理的序列长度”放在 DP 状态中不太好转移，我们还可以给状态再加一维，把 b 中的最后一个数也记录下来。设 $f_{i,j}$ 表示完成了前 i 个数的构造，且 $b_i = j$ 时，S 的最小值。

$$f_{i,j} = \min_{0 \le k < j} \{f_{i-1,k} + |a_i - j|\}$$

我们只需要预先对 a 进行离散化操作，即可把 j 的范围变为 $O(n)$。同时，本题的转移与上一道例题 LCIS 非常相似，决策集合只多不少，可以实现 $O(1)$ 的转移。时间复杂度为 $O(n^2)$。

Making the Grade

既然只把“已经处理的序列长度”放在 DP 状态中不太好转移，我们还可以给状态再加一维，把 b 中的最后一个数也记录下来。设 $f_{i,j}$ 表示完成了前 i 个数的构造，且 $b_i = j$ 时，S 的最小值。

$$f_{i,j} = \min_{0 \le k < j} \{f_{i-1,k} + |a_i - j|\}$$

我们只需要预先对 a 进行离散化操作，即可把 j 的范围变为 $O(n)$。同时，本题的转移与上一道例题 LCIS 非常相似，决策集合只多不少，可以实现 $O(1)$ 的转移。时间复杂度为 $O(n^2)$。

思考题：把一个序列 a 变为非严格单调递增的（单调不下降的），至少需要修改多少个数？

Making the Grade

既然只把“已经处理的序列长度”放在 DP 状态中不太好转移，我们还可以给状态再加一维，把 b 中的最后一个数也记录下来。设 $f_{i,j}$ 表示完成了前 i 个数的构造，且 $b_i = j$ 时，S 的最小值。

$$f_{i,j} = \min_{0 \le k < j} \{f_{i-1,k} + |a_i - j|\}$$

我们只需要预先对 a 进行离散化操作，即可把 j 的范围变为 $O(n)$。同时，本题的转移与上一道例题 LCIS 非常相似，决策集合只多不少，可以实现 $O(1)$ 的转移。时间复杂度为 $O(n^2)$。

思考题：把一个序列 a 变为非严格单调递增的（单调不下降的），至少需要修改多少个数？
答案为序列 a 的总长度减去 a 的最长不下降子序列的长度。

Making the Grade

既然只把“已经处理的序列长度”放在 DP 状态中不太好转移，我们还可以给状态再加一维，把 b 中的最后一个数也记录下来。设 $f_{i,j}$ 表示完成了前 i 个数的构造，且 $b_i = j$ 时，S 的最小值。

$$f_{i,j} = \min_{0 \le k < j} \{f_{i-1,k} + |a_i - j|\}$$

我们只需要预先对 a 进行离散化操作，即可把 j 的范围变为 $O(n)$。同时，本题的转移与上一道例题 LCIS 非常相似，决策集合只多不少，可以实现 $O(1)$ 的转移。时间复杂度为 $O(n^2)$。

思考题：把一个序列 a 变为非严格单调递增的（单调不下降的），至少需要修改多少个数？
答案为序列 a 的总长度减去 a 的最长不下降子序列的长度。
把一个序列 a 变为严格单调递增的，至少需要修改多少个数？

Making the Grade

既然只把“已经处理的序列长度”放在 DP 状态中不太好转移，我们还可以给状态再加一维，把 b 中的最后一个数也记录下来。设 $f_{i,j}$ 表示完成了前 i 个数的构造，且 $b_i = j$ 时，S 的最小值。

$$f_{i,j} = \min_{0 \le k < j} \{f_{i-1,k} + |a_i - j|\}$$

我们只需要预先对 a 进行离散化操作，即可把 j 的范围变为 $O(n)$。同时，本题的转移与上一道例题 LCIS 非常相似，决策集合只多不少，可以实现 $O(1)$ 的转移。时间复杂度为 $O(n^2)$。

思考题：把一个序列 a 变为非严格单调递增的（单调不下降的），至少需要修改多少个数？
答案为序列 a 的总长度减去 a 的最长不下降子序列的长度。
把一个序列 a 变为严格单调递增的，至少需要修改多少个数？
构造序列 $b_i = a_i - i$，就变成了上一个问题。答案为总长度减去 b 的最长不下降子序列的长度。

例题

Mobile Service

一个公司有三个移动服务员，最初分别在位置 $1, 2, 3$ 处。
如果某个位置（用一个整数表示）有一个请求，那么公司必须派出某名员工赶到那个地方去。某一时刻只有一个员工能移动，且不允许在同样的位置出现两名员工。从 p 到 q 移动一名员工，需要花费 $c(p, q)$。这个函数不一定对称，但保证 $c(p, p) = 0$。
给出 n 个请求，请求发生的位置分别为 $p_1 \sim p_n$。公司必须按照顺序依次满足所有请求，目标是最小化公司花费，请你帮忙计算这个最小花费。$n \leq 1000$，位置是 $1 \sim 200$ 的整数。

例题

Mobile Service

一个公司有三个移动服务员，最初分别在位置 $1, 2, 3$ 处。
如果某个位置（用一个整数表示）有一个请求，那么公司必须派出某名员工赶到那个地方去。某一时刻只有一个员工能移动，且不允许在同样的位置出现两名员工。从 p 到 q 移动一名员工，需要花费 $c(p, q)$。这个函数不一定对称，但保证 $c(p, p) = 0$。
给出 n 个请求，请求发生的位置分别为 $p_1 \sim p_n$。公司必须按照顺序依次满足所有请求，目标是最小化公司花费，请你帮忙计算这个最小花费。$n \leq 1000$，位置是 $1 \sim 200$ 的整数。

容易发现，DP 的阶段就是"已经完成的请求数量"，通过指派一名服务员，可以把一个"完成 $i-1$ 个请求"的状态转移到"完成 i 个请求的状态"。
为了计算指派服务员的花费，就必须要知道状态转移时每个服务员的位置。最直接的想法就是把三个服务员的位置也放在 DP 的状态中。
设 $f[i, x, y, z]$ 表示完成了前 i 个请求，三个员工分别位于 x, y, z 时，公司的最小花费。

Mobile Service

考虑 $f[i,x,y,z]$ 能够更新哪些状态，转移显然有三种，就是指派哪个员工去 p_{i+1} 处。这里只列出第一种。

$$f[i+1,p_{i+1},y,z]=min(f[i+1,p_{i+1},y,z],f[i,x,y,z]+c(x,p_{i+1}))$$

题目要求同一位置不能出现两个员工，注意判断状态的合法性。

Mobile Service

考虑 $f[i,x,y,z]$ 能够更新哪些状态，转移显然有三种，就是指派哪个员工去 p_{i+1} 处。这里只列出第一种。

$$f[i+1,p_{i+1},y,z]=min(f[i+1,p_{i+1},y,z],f[i,x,y,z]+c(x,p_{i+1}))$$

题目要求同一位置不能出现两个员工，注意判断状态的合法性。
该算法的规模大约在 $1000*200^3$，不能承受。通过观察我们可以发现，在完成了第 i 个请求之后，一定有某个员工位于位置 p_i，所以一个状态只需要记录阶段 i 和另外两个员工的位置即可。
于是我们可以用 $f[i,x,y]$ 表示完成了前 i 个请求，一个员工位于 p_i，另外两位员工分别位于 x,y 的最小花费。同样是三种转移，分别让三名员工去 p_{i+1} 处。设 $p_0=3$，初值即为 $f[0,1,2]=0$，目标为 $f[n,?,?]$。

在设计 DP 的状态时，可以先划分阶段，如果仅凭“阶段”不足以表示状态，可以添加附加信息作为新的维度来表示状态。
在状态转移时，若总是从一个阶段转移到下一个阶段，我们就不需要考虑附加信息的枚举顺序。
在确定 DP 状态时，可以思考一下是否能用更简略的信息来表示状态。如果某些维度表示的信息可以互相推导得出，则可以删除掉冗余维度。

例题

传纸条

给定一个 $n*m$ 的矩阵 A，每个格子中有一个整数。现在需要找到两条从左上角 $(1,1)$ 到右下角 (n,m) 的路径，路径上的每一步只能向右或者向下走。路径经过的格子中的数会被取走，如果两条路径经过同一个格子，只算一次。求取得的数之和最大是多少。$n,m \leq 50$。

例题

传纸条

给定一个 $n*m$ 的矩阵 A，每个格子中有一个整数。现在需要找到两条从左上角 $(1,1)$ 到右下角 (n,m) 的路径，路径上的每一步只能向右或者向下走。路径经过的格子中的数会被取走，如果两条路径经过同一个格子，只算一次。求取得的数之和最大是多少。$n,m \le 50$。

首先尝试寻找一个线性的“阶段”。考虑路径形成的过程，由于每次只能向右或者向下走，因此如果两条路径走到了同一个格子，那么它们目前的“路径长度”肯定是相同的。因此我们可以把当前走过的步数作为 DP 的“阶段”。对于每个阶段，我们把两条路径格子拓展一步，路径长度增加一，转移到下个阶段。

传纸条

接着考虑除了路径长度，我们还需要哪些信息。显而易见的是，每次拓展肯定从路径末尾开始拓展，所以我们还需要记录两条路径的末尾位置 $(x_1, y_1), (x_2, y_2)$。我们当然可以增加四个维度，但根据上一道题的启发，设路径长度为 i，我们发现有下列等式：

$$x_1 + y_1 = x_2 + y_2 = i + 2$$

所以 i, x_1, x_2 就可以描述一个状态了。设 $f[i, x_1, x_2]$ 表示两条路径长度均为 i，第一条路径末尾在第 x_1 行，第二条路径末尾在第 x_2 行时，已经取出的数的最大值。

初值为 $f[0, 1, 1] = A[1, 1]$，目标为 $f[n + m - 2, n, n]$。每条路径都有向下、向右两种拓展方法，故共有 $2 * 2 = 4$ 种转移。以两条路径均往右走为例，当 $x_1 = x_2$ 时，走到同一个格子：

$$f[i + 1, x_1, x_2] = \max(f[i + 1, x_1, x_2], f[i, x_1, x_2] + A[x_1, y_1 + 1])$$

当 $x_1 \neq x_2$ 时：

$$f[i+1, x_1, x_2] = \max(f[i+1, x_1, x_2], f[i, x_1, x_2]+A[x_1, y_1+1]+A[x_2, y_2+1])$$

例题

I-country

在 $n * m$ 的矩阵中，每个格子有一个权值。要求寻找一个包含 k 个格子的凸联通块（联通块中间没有空缺，并且轮廓是凸的），使这个连通块中的格子的权值和最大。求出这个最大的权值和，并给出连通块的具体方案。

$n, m \leq 15, k \leq 225$。

例题

I-country

在 $n*m$ 的矩阵中，每个格子有一个权值。要求寻找一个包含 k 个格子的凸联通块（联通块中间没有空缺，并且轮廓是凸的），使这个连通块中的格子的权值和最大。求出这个最大的权值和，并给出连通块的具体方案。
$n, m \le 15, k \le 225$。

任何一个凸连通块可以划分为连续的若干行，每行的左端点列号先递减、后递增，右端点列号先递增、后递减。我们可以依次考虑从 $n*m$ 矩阵的每一行中选择哪些格子来构成所求的凸连通块。需要关注的信息如下：

- 当前已经处理完的行数。
- 已经选出的格子数。
- 当前行已选格子的左端、右端位置。
- 当前行左侧轮廓的单调性类型（递增还是递减）
- 当前行右侧轮廓的单调性类型（递增还是递减）

I-country

所以我们用这些需要的信息来表示状态，设 $f[i,j,l,r,x,y]$ 表示前 i 行总共选了 j 个格子，第 i 行选择了第 l 个到第 r 个格子（若不选则均为 0），左侧单调性为 x，右侧单调性为 y （0 表示递增，1 表示递减）时，能构成的凸连通块的最大权值和。
设 $m=\sum_{p=l}^{r} A[i,p]$，来考虑一下转移方程。当左边界列号递减，右边界列号递增时（两个边界都在扩张）:

$$f[i,j,l,r,1,0]=m+\begin{cases} f[i-1,0,0,0,1,0](j=r-l+1) \\ \max\limits_{l\le p\le q\le r}\{f[i-1,j-(r-l+1),p,q,1,0]\}(j>r-l+1) \end{cases}$$

其它三种情况的转移方程同理，额外注意一下 x,y 的取值即可。
初值为 $f[i,0,0,0,1,0]=0$，目标为 $\max\{f[i,K,l,r,x,y]\}$，时间复杂度为 $O(nm^4K)=O(n^2m^5)$。
本题还需要输出方案，再另外用一个数组来记录每个 f 的最大值是从哪里转移过来的即可。

例题

Cookies

圣诞老人有 m 个饼干，准备全部分给 n 个孩子。每个孩子有一个贪婪度，第 i 个孩子的贪婪度为 g_i。如果有 a_i 个孩子拿到的饼干比第 i 个孩子多，那么第 i 个孩子会产生 $g_i * a_i$ 的怨气。
给定 n, m 和序列 g，请你安排一种分配方式，使得每个孩子至少分到一块饼干，并且所有孩子的怨气之和最小。$1 \le n \le 30, n \le m \le 5000$。

例题

Cookies

圣诞老人有 m 个饼干，准备全部分给 n 个孩子。每个孩子有一个贪婪度，第 i 个孩子的贪婪度为 g_i。如果有 a_i 个孩子拿到的饼干比第 i 个孩子多，那么第 i 个孩子会产生 $g_i * a_i$ 的怨气。
给定 n, m 和序列 g，请你安排一种分配方式，使得每个孩子至少分到一块饼干，并且所有孩子的怨气之和最小。$1 \le n \le 30, n \le m \le 5000$。

在这道题目中，“已经获得饼干的孩子数”和“已经发放的饼干数”显然应该作为 DP 的阶段。不过，一个孩子的怨气大小与其它孩子获得的饼干数有关，这让我们很难对状态划分出“子结构”，同时也很难计算每个孩子的怨气值。
但仔细思考可以发现，贪婪度大的孩子应该获得更多的饼干。可以使用邻项交换法来证明这一点。因此，可以把 n 个孩子按照贪婪度从大到小排序，这样他们分配到的饼干数是单调递减的。

Cookies

这样我们就可以设计状态了。设 $f[i,j]$ 表示前 i 个孩子一共分配 j 块饼干时，怨气和的最小值。直观的思想时考虑给第 $i+1$ 个孩子多少块饼干，然后进行转移。转移时有两种情况：

- 第 $i+1$ 个孩子获得的饼干数比第 i 个孩子少，此时 $a_{i+1}=i$。
- 第 $i+1$ 个孩子获得的饼干数与第 i 个孩子相同，此时还需要知道 i 前面有几个孩子与第 i 个获得的饼干数也相同，才能算出 a_{i+1}。

我们发现不管哪种情况，都只有知道了第 i 个孩子获得的饼干数，才能往后转移。但如果我们给 DP 添加一维，复杂度又太高了。
实际上，在碰到此类问题，要求某个序列单减或者单增时，我们可以把原序列看成若干个前缀连续 1 或者后缀连续 1 的序列之和。在本题中，如果 a 序列为 $\{4,3,3,1\}$，我们可以把它看成 $\{1,0,0,0\}$，$\{1,1,1,0\}$，$\{1,1,1,0\}$，$\{1,1,1,1\}$ 四个序列的和。

Cookies

实际操作过程中，若第 i 个孩子获得的饼干数大于 1，则等价于分配 $j-i$ 块饼干给前 i 个孩子，每人少拿一块饼干，获得的饼干数的相对大小不变，怨气之和也不变。如果第 i 个孩子只拿了一块饼干，则枚举 i 前面有多少个孩子也获得 1 块饼干。
转移方程如下：

$$f[i,j] = \min \begin{cases} f[i,j-i] \\ \min\limits_{0 \le k < i} \{f[k, j-(i-k)] + k * \sum\limits_{p=k+1}^{i} g_p\} \end{cases}$$

初始状态为 $f[0,0]=0$，目标为 $f[n,m]$，时间复杂度为 $O(n^2m)$。
在本题中，我们先通过贪心策略，在 DP 前对 n 个孩子进行排序，才能确定 DP 状态的计算顺序，划分阶段。同时我们还利用的相对大小的不变性，把第 $i+1$ 个孩子获得的饼干数先缩放到 1，再考虑前面有多少个孩子获得的饼干数相等，极大简化了需要计算的问题。