算法竞赛进阶指南
《Cookies》解析

河南省实验 程远洋

SYOI 1232. Cookies

算法1：搜索算法（DFS + 剪枝）

算法思想

采用深度优先搜索枚举所有可能的分配方案，并通过多个剪枝策略优化：

排序剪枝：将孩子按贪婪度降序排序，只搜索非递增的分配序列

边界剪枝：保证每个孩子至少1块饼干，后续孩子也至少能分到1块

最优性剪枝：当当前方案的怨气已超过已知最优解时停止搜索

对称性剪枝：对于相同饼干数的孩子，他们的相对顺序不影响怨气计算

算法复杂度

- **时间复杂度**：最坏情况 $O(m^n)$，但实际通过剪枝会大大减少搜索空间
- **空间复杂度**：$O(n)$

n	4
m	9

i	1	2	3	4
g[i]	2	1	5	8

i	1	2	3	4
idx[i]	1	2	3	4

i	1	2	3	4
idx[i]	4	3	1	2

i	1	2	3	4
g[i]	8	5	2	1

排序剪枝（核心优化）

原理：存在一个最优解，使得分配方案中饼干数量与贪婪度同序（非递增）。

数学证明：对于任意两个孩子 i 和 j，如果 $g[i] > g[j]$ 但 $c[i] < c[j]$（c 为饼干数），则交换 $c[i]$ 和 $c[j]$ 不会增加总怨气：

交换前孩子 i 怨气：$g[i] \times a_i$

交换前孩子 j 怨气：$g[j] \times a_j$

交换后孩子 i 怨气：$g[i] \times a_j$

交换后孩子 j 怨气：$g[j] \times a_i$

怨气变化：$(g[i] - g[j]) \times (a_j - a_i)$

i	1	2	3	4
g[i]	2	1	5	8
c1[i]	2	3	1	3
c2[i]	3	2	1	3

2*2 + 0*1 + 3*5 + 0*8

0*2 + 2*1 + 3*5 + 0*8

由于 $g[i] > g[j]$ 且 $c[i] < c[j]$，有 $a_i \geq a_j$（饼干少的孩子被更多人超过），所以 $(a_j - a_i) \leq 0$，总怨气不会增加。

实现：将孩子按贪婪度降序排序，搜索时只考虑非递增的分配序列。

搜索空间定义

设排序后孩子为 $1, 2, \ldots, n$，贪婪度 $g[1] \geq g[2] \geq \cdots \geq g[n]$。搜索分配序列 $c[1..n]$ 满足：

$c[1] \geq c[2] \geq \cdots \geq c[n]$（非递增）
$\sum_{i=1}^{n} c[i] = m$
$c[i] \geq 1$（每人至少一块）

适用数据范围：

- 推荐：$n \leq 10, m \leq 30$
- 勉强：$n \leq 15, m \leq 50$

递归函数设计

```cpp
void dfs(int pos,     // 当前正在分配的孩子位置（排序后）
         int remain,  // 剩余饼干数
         long long anger)  // 当前已产生的怨气
```

枚举策略

对于当前孩子 pos，枚举可能的饼干数 val：

- **上限**：$\min(c[pos-1], \text{remain} - (n - pos))$
  - $c[pos-1]$：保证非递增
  - $\text{remain} - (n - pos)$：给后面的孩子每人至少留一块饼干
- **下限**：1（每人至少一块）

步骤1：预处理

输入 n, m 和贪婪度数组 $g[1..n]$

按贪婪度降序排序孩子，记录原始索引映射

重新排列贪婪度数组为降序

步骤3：结果输出

输出最小怨气 `min_anger`

将最优方案映射回原始孩子顺序输出

程序填空见备注

步骤2：深度优先搜索

```
dfs(1, m, 0):
    如果 pos > n:
        如果 remain == 0 且 anger < min_anger:
            更新 min_anger 和最优分配方案
        返回

    计算当前孩子饼干数上限 max_val
    对于 val = max_val 递减到 1:
        计算新怨气 new_anger
        如果 new_anger >= min_anger: 剪枝
        递归调用 dfs(pos+1, remain-val, new_anger)
```

n	4
m	8

i	1	2	3	4
g[i]	2	10	15	8

i	3	2	4	1
ans[i]	2	2	2	2

n	4
m	10

i	1	2	3	4
g[i]	2	10	15	8

i	3	2	4	1
ans[i]	3	3	3	1

n	4
m	9

i	1	2	3	4
g[i]	2	1	5	8

i	4	3	1	2
ans[i]	3	3	2	1

n	6
m	13

i	1	2	3	4	5	6
g[i]	7	7	7	1	1	1

i	1	2	3	4	5	6
ans[i]	3	3	3	2	1	1

动态规划算法思想详解

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i	1	2	3	4
ans[i]	3	3	2	1

一、算法总体框架

这是一个典型的**基于排序和状态转移的动态规划**问题。核心思想是将孩子按贪婪度降序排序，然后通过巧妙的DP状态设计，避免直接枚举每个孩子的饼干数，而是考虑"整体加一"和"新增层级"两种操作。

二、阶段划分

阶段定义

阶段：按贪婪度降序排序后，考虑前 `i` 个孩子的分配情况，其中 `i` 从 `0` 到 `n`。

阶段含义

- 阶段 `i` 表示我们已经处理了前 `i` 个孩子（贪婪度最大的前 `i` 个孩子）

f(3,6)

2 2 2 0

f(3,9)

3 3 3 0

f(2,8)+2*2

4 4 1 4

f(1,7)+1*(5+2)

7 1 1 7

三、状态设计

状态定义

状态：`dp[i][j]` 表示 **前 `i` 个孩子（按贪婪度降序排列）分配 `j` 块饼干的最小怨气总和。**

状态参数

`i`：已经考虑的孩子数量（$0 \le i \le n$）

`j`：已经分配的饼干总数（$0 \le j \le m$）

四、决策集合

决策类型1：整体加一

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i	1	2	3	4
ans[i]	3	3	2	1

决策内容：给前 `i` 个孩子每人加一块饼干。

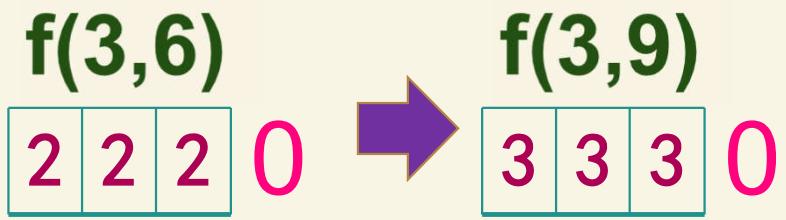
适用条件：当所有前 `i` 个孩子的饼干数都至少为2时。

状态转移：

```
dp[i][j] = dp[i][j-i]   (当 j ≥ i 时)
```

决策依据：

- 如果所有孩子饼干数都≥2，那么每人减1块饼干，相对大小不变
- 怨气值不变，饼干总数减少 `i`
- 这是一个**等价变换**，不改变怨气值



n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i	1	2	3	4
ans[i]	3	3	2	1

决策类型2：新增层级

决策内容：让最后 `len` 个孩子（第 `i-len+1` 到第 `i` 个）饼干数为1。

适用条件：存在一个 `len`（1 ≤ len ≤ i），使得第 `i-len+1` 到第 `i` 个孩子饼干数为1，而前 `i-len` 个孩子饼干数>1。

状态转移：

```
dp[i][j] = min{ dp[i-len][j-len] + (i-len) × (g[i-len+1] + ... + g[i]) }
```

其中 1 ≤ len ≤ i，且 j ≥ len

f(3,9)

f(2,8)+2*2 → 4 4 1 → 4

f(1,7)+1*(5+2) → 7 1 1 → 7

五、状态转移方程

完整方程

```
dp[i][j] = min{
    // 决策1：整体加一（如果j ≥ i）
    dp[i][j-i],

    // 决策2：新增层级（枚举所有可能的len）
    min{ dp[i-len][j-len] + (i-len) × sum(i-len+1, i) }
        其中 1 ≤ len ≤ i，且 j ≥ len
}
```

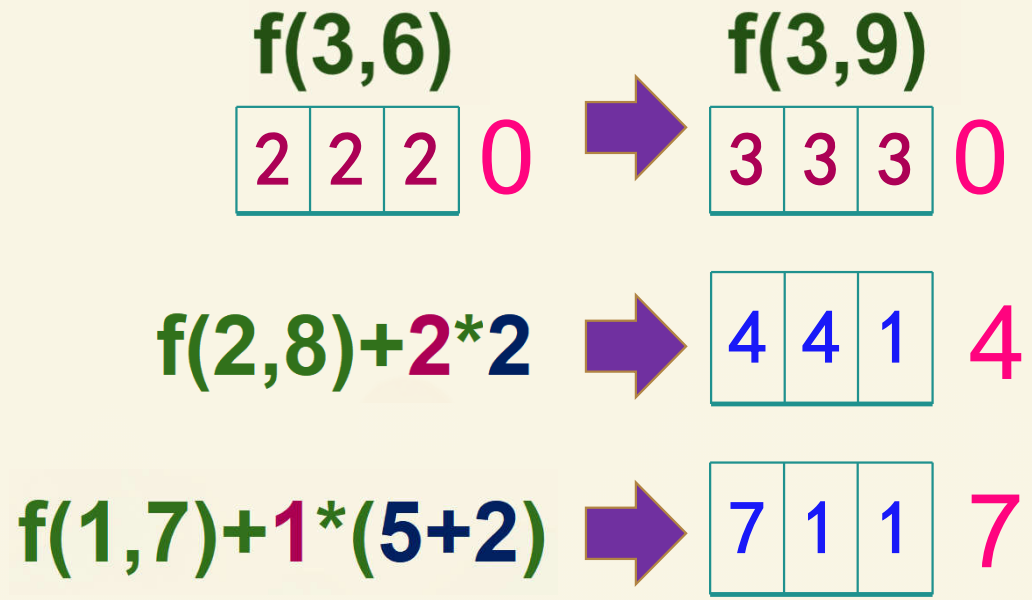
边界条件

```
dp[0][0] = 0    // 0个孩子，0块饼干，怨气为0
dp[0][j] = INF (j > 0)   // 不可能的情况
dp[i][j] = INF (j < i)    // 饼干不够每人一块
```

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i	1	2	3	4
ans[i]	3	3	2	1



n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i	1	2	3	4
ans[i]	3	3	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	**0**	0	0	0	0	0	0	0
2	INF	INF	0	**5**	0	5	0	5	0	5
3	INF	INF	INF	0	**7**	4	0	4	4	0
4	INF	**INF**	INF	INF	0	**8**	6	3	0	7

f(4,5) = min

- **f(3,4) + 3 * 1** — 2 1 1 1 — 10
- **f(2,3) + 2 * (2+1)** — 2 1 1 1 — 11
- **f(1,2) + 1 * (5+2+1)** — 2 1 1 1 — 8

i \ j	0	1	2	3	4	5	6	7	8	9
0	0	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	0	5	0	5	0	5	0	5
3	INF	INF	INF	0	7	4	0	4	4	0
4	INF	INF	INF	INF	0	8	6	3	0	7

f(3,4) + 3 * 1 | 2 1 1 1 | 10

f(2,3) + 2 * (2+1) | 2 1 1 1 | 11

f(1,2) + 1 * (5+2+1) | 2 1 1 1 | 8

在动态规划算法中，“模式假设”指的是我们在设计状态转移方程时，为了覆盖所有可能的分配方案，而**人为设想的一种特定分配结构**，并基于这种结构计算怨气增量。这种假设并非最终方案的实际结构，但通过枚举所有可能的假设参数，可以确保不遗漏任何最优解。

新增层级模式

假设：存在某个整数 k $(0 \le k < i)$ ，使得：

- 前 k 个孩子的饼干数**都大于 1**。
- 后 $i-k$ 个孩子的饼干数**都等于 1**。

怨气增量：后 $i-k$ 个孩子中，每人前面有 k 个饼干数多于他们的孩子，因此总怨气增加 $k \times \sum_{t=k+1}^{i} g[t]$。

转移：$f(i,j) = f(k, j-(i-k)) + k \times (\text{sum}[i] - \text{sum}[k])$。

为什么模式假设能保证正确性?

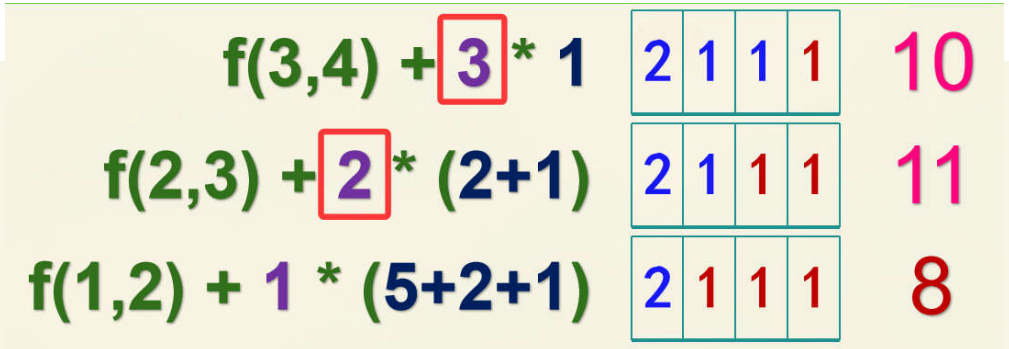
虽然实际最优分配方案可能不完全符合上述假设（例如，前 k 个孩子中可能有饼干数等于 1 的，或后 $i-k$ 个孩子中可能有饼干数大于 1 的），但动态规划通过**枚举所有可能的** k，确保了所有情况都被覆盖。原因如下：

完备性：任何非递增的分配序列 $c[1] \geq c[2] \geq \cdots \geq c[i] \geq 1$ 都可以通过以下步骤生成：

- 重复执行"新增层级"操作，确定哪些位置饼干数为 1。
- 重复执行"整体加一"操作，增加非 1 位置的饼干数。

因此，通过枚举 k（即最后一个饼干数为 1 的连续段的起始位置），并配合整体加一操作，可以构造出任意合法分配。

最优子结构：如果某个最优方案不完全符合某个 k 对应的模式，那么一定存在另一个 k' 使得转移值更小（即更接近实际怨气）。例如，若前 k 个孩子中有人饼干数等于 1，则实际上后 $i-k$ 个孩子前面饼干数多于他们的人数少于 k，此时用 k 计算会高估怨气，但动态规划会通过枚举更小的 k' 得到更小的怨气值，从而保证最终取到最小值。



n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i	1	2	3	4
ans[i]	3	3	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	**0**	0	0	0
2	INF	INF	0	5	0	5	0	**5**	0	5
3	INF	INF	INF	0	7	4	0	4	**4**	0
4	INF	INF	INF	INF	0	**8**	6	3	0	**7**

f(4,9) = min

- **f(4,5)整体加一** 2 1 1 1 → 3 2 2 2 — 8
- **f(3,8) + 3 * 1** — 3 3 2 1 — 7
- **f(2,7) + 2 * (2+1)** — 4 3 1 1 — 11
- **f(1,6) + 1 * (5+2+1)** — 6 1 1 1 — 8

六、算法正确性证明

定理1：排序不变性

在最优解中，贪婪度大的孩子饼干数不少于贪婪度小的孩子。 **证明**：交换论证法。如果存在逆序，交换不会使解变差。

定理2：决策完备性

任意一个最优分配方案都可以通过一系列"新增层级"和"整体加一"操作得到。 **证明**：

从空开始，不断"新增层级"（给一批孩子分配1块饼干）
然后通过"整体加一"调整饼干数
这两种操作的任意组合可以生成所有可能的非递增分配序列

定理3：最优子结构

前 `i` 个孩子的最优分配，其子结构也是最优的。 **证明**：如果子结构不是最优，可以用更优的替换，得到整体更优解，矛盾。

七、方案构造方法

记录决策

对于每个状态 `(i, j)`，记录：

决策类型：整体加一 或 新增层级

参数：如果是新增层级，记录 `len` 值

前驱状态：从哪个状态转移而来

回溯构造

从最终状态 `(n, m)` 开始：

如果是**整体加一**：给前 `i` 个孩子每人加一块饼干

如果是**新增层级**：给最后 `len` 个孩子分配1块饼干

回溯到前驱状态，重复直到 `(0, 0)`

构造过程示例

状态序列：`(n,m) → (i1,j1) → ... → (0,0)`

操作序列：对应决策的逆操作

最终分配：基础值（由新增层级决定）+ 累加值（由整体加一决定）

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	**0**	5	**0**	5	0	5	0	5
3	INF	INF	INF	0	7	**4**	0	4	**4**	0
4	INF	INF	INF	INF	0	8	6	3	0	**7**

i	1	2	3	4
add[i]				

i	1	2	3	4
base[i]				

3	3	2	1

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

j / i	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	**0**	5	**0**	5	0	5	0	5
3	INF	INF	INF	0	7	**4**	0	4	**4**	0
4	INF	INF	INF	INF	0	8	6	3	0	**7**

i	1	2	3	4
add[i]				

i	1	2	3	4
base[i]				1

2	2	1

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	**0**	5	**0**	5	0	5	0	5
3	INF	INF	INF	0	7	**4**	0	4	**4**	0
4	INF	INF	INF	INF	0	8	6	3	0	**7**

i	1	2	3	4
add[i]	1	1	1	

i	1	2	3	4
base[i]				1

2	2	1

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	**0**	5	**0**	5	0	5	0	5
3	INF	INF	INF	0	7	**4**	0	4	**4**	0
4	INF	INF	INF	INF	0	8	6	3	0	**7**

i	1	2	3	4
add[i]	1	1	1	

i	1	2	3	4
base[i]			1	1

1	1

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	**0**	5	**0**	5	0	5	0	5
3	INF	INF	INF	0	7	**4**	0	4	**4**	0
4	INF	INF	INF	INF	0	8	6	3	0	**7**

i	1	2	3	4
add[i]	2	2	1	

i	1	2	3	4
base[i]			1	1

1	1

n	4
m	9

i	1	2	3	4
g[i]	8	5	2	1

i \ j	0	1	2	3	4	5	6	7	8	9
0	**0**	INF	INF	INF	INF	INF	INF	INF	INF	INF
1	INF	0	0	0	0	0	0	0	0	0
2	INF	INF	**0**	5	**0**	5	0	5	0	5
3	INF	INF	INF	0	7	**4**	0	4	**4**	0
4	INF	INF	INF	INF	0	8	6	3	0	**7**

i	1	2	3	4
add[i]	2	2	1	

+

i	1	2	3	4
base[i]	1	1	1	1

i	1	2	3	4
ans[i]	3	3	2	1

八、时间复杂度分析

状态数量

- `i` 从 0 到 `n`：共 `n+1` 种
- `j` 从 0 到 `m`：共 `m+1` 种
- 总状态数：`O(n × m)`

每个状态的决策数

决策1：整体加一，`O(1)` 时间

决策2：新增层级，需要枚举 `len` 从 1 到 `i`，`O(n)` 时间

总决策复杂度：`O(n)`

总时间复杂度

- 状态数 × 每个状态的决策数 = `O(n × m) × O(n) = O(n² × m)`
- 对于 `n ≤ 30, m ≤ 5000`：`30² × 5000 = 4,500,000`，可接受

空间复杂度

- DP 表：`O(n × m)`
- 路径记录：`O(n × m)`
- 总空间：`O(n × m)`，可优化但需记录路径