鸡蛋掉落

王曦 2021192010

数学与统计学院

算法设计与分析

2023 年 04 月 25 日

1. 实验内容

实验内容

1. 给出鸡蛋掉落问题的 DP 方程.
2. 随机产生 f 和 e 值, 对小数据用蛮力法验证算法的正确性.
3. 随机产生 f 和 e 值, 对不同数据规模测试算法效率, 与理论效率对比, 提供在有限时间内能处理的数据的最大规模.
4. 该算法的时间效率和空间效率是否有提升空间.

- 题意: 有 m 个相同的鸡蛋, 用从低到高分别为 $1 \sim n$ 的 n 层楼实验. 已知存在分界楼层 $f \in [0, n]$ $s.t.$ 鸡蛋从任意不高于 f 的楼层落下都不会打破. 每次操作可取一个未打破的鸡蛋将其从任一楼层 $x \in [1, n]$ 落下, 若鸡蛋打破, 则丢弃; 否则可重复使用该鸡蛋. 求确定 f 所需的最小实验次数.

2. 实验环境与约定

实验环境与约定

- 实验环境: GNU C++17 (O2).
- 约定所有实际运行时间都不包含数据生成和输入的时间.

3. 暴力法

暴力法

- 思路: $dfs(n,m)$ 表示用 m 个鸡蛋确定 n 层楼的分界楼层所需的最小实验次数.

(1) 递归终止条件:

(i) $n=1$, 即只有一层楼时, 无论有多少个鸡蛋, 所需的最小实验次数都为 1.

(ii) $m=1$, 即只有一个鸡蛋时, 无论有多少层楼, 最坏情况都需一层一层实验, 则所需的最小试验次数为 n.

(2) 按第一个鸡蛋在楼层 k 落下的情况分类:

(i) 鸡蛋碎, 则分界楼层 $f \in [1, k-1]$, 需用剩下的 $(m–1)$ 个鸡蛋确定 $(k–1)$ 层楼的分界楼层, 所需的最小实验次数为 $dfs(k–1, m-1)$.

(ii) 鸡蛋未碎, 则分界楼层 $f \in [k, n]$, 需用剩下的 m 个鸡蛋确定 $(n–k)$ 层楼的分界楼层, 所需的最小实验次数为 $dfs(n–k, m)$.

(3) 递推式:

$$dfs(n,m) = \min_{1 \le k \le n} \{dfs(k-1, m-1), dfs(n-k, m)\} + 1.$$

小数据验证

求得 $n \in [1,6], m \in [1,6]$ 的答案, 如下表所示:

$n \setminus m$	1	2	3	4	5	6
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	2	2	2	2	2
4	4	3	3	3	3	3
5	5	3	3	3	3	3
6	6	3	3	3	3	3

4. DP (朴素)

DP (朴素)

- 状态设计: $dp[i][j]$ 表示用 j 个鸡蛋找到 i 层的分界楼层 f 所需的最小实验次数.
- 初始条件: $dp[i][j] = \begin{cases} i, j = 1 \\ INF, 2 \le j \le m \end{cases} \quad (1 \le i \le n).$
- 最终答案: $ans = dp[n][m]$.

DP 的状态转移方程——对集合的划分

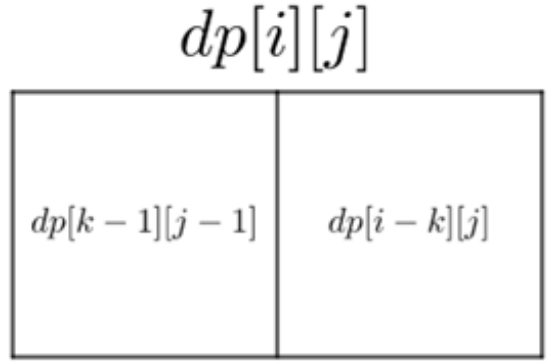

Figure 1: 集合示意图

- 如上图, 大矩形为所有 $dp[i][j]$ 表示的方案构成的集合.
- 因每个方案都至少落下一个鸡蛋, 设第一个鸡蛋在第 k 层落下. 按第一个鸡蛋的结果对集合进行划分.

(1) 若第一个鸡蛋打破 (左半边的小矩形), 则 $f \in [1, k-1]$, 还需用剩下的 $(j-1)$ 个鸡蛋确定 $1 \sim (k-1)$ 层中的分界楼层, 所需的最小实验次数为 $dp[k-1][j-1]+1$. 枚举所有 $k \in [1, n]$, 则最小实验次数为所有 k 对应的最小实验次数的最小值, 故状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{dp[k-1][j-1]\} + 1.$$

DP 的状态转移方程——对集合的划分

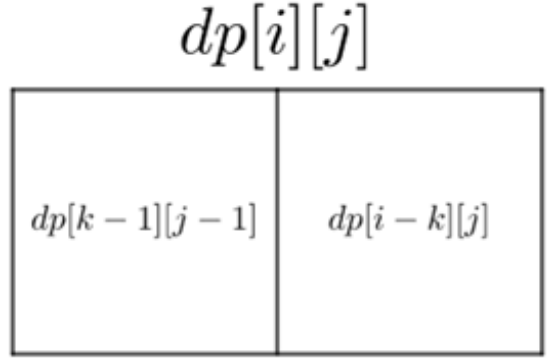

Figure 2: 集合示意图

(2) 若第一个鸡蛋未打破 (右半边的小矩形), 则 $f \in [k, n]$, 还需用剩下的 j 个鸡蛋确定 $(k+1) \sim n$ 层中的分界楼层, 所需的最小实验次数为 $dp[i-k][j]+1$. 枚举所有 $k \in [1, n]$, 则最小实验次数为所有 k 对应的最小实验次数的最小值, 故状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{dp[i-k][j]\} + 1.$$

因最小实验次数为最坏情况下的实验次数, 故状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

时空复杂度

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

- 状态数 $O(nm)$, 转移 $O(n)$, 总时间复杂度 $O(n^2m)$.
- 总空间复杂度 $O(nm)$.
- 求得 $n \in [1,6], m \in [1,6]$ 的答案, 如下表所示:

$n \setminus m$	1	2	3	4	5	6
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	2	2	2	2	2
4	4	3	3	3	3	3
5	5	3	3	3	3	3
6	6	3	3	3	3	3

与暴力法求得的答案相同.

DP (朴素) 的实际运行效率

- 运行效率与楼层数 n 的关系: 固定鸡蛋数 $m = 10$, 分别取楼层数 $n = 100, ..., 1000$, 统计运行时间.
- 以 $n = 500, m = 10$ 为基准, 计算理论运行时间.

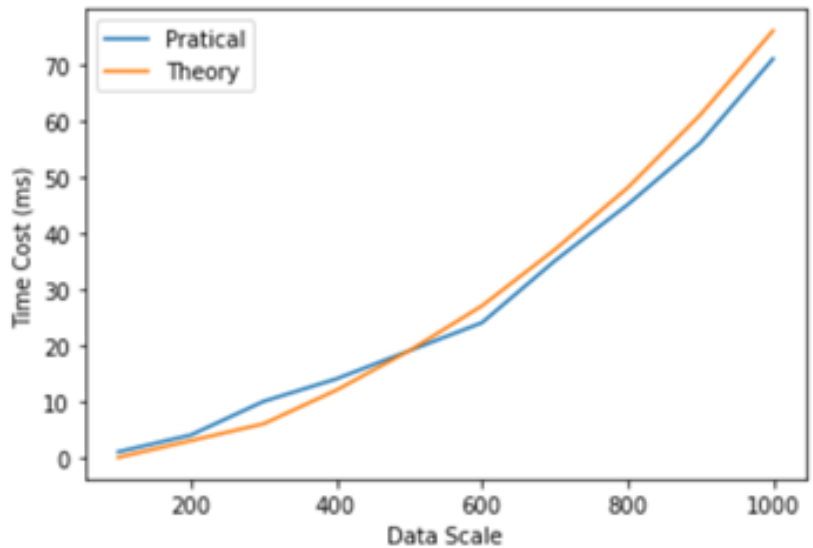

Figure 3: 实际运行时间与理论运行时间随楼层数 n 的变化关系

- 分析: 实际运行时间与楼层数 n 成抛物线关系.

DP (朴素) 的实际运行效率

- 运行效率与鸡蛋数 m 的关系: 固定楼层数 $n = 1000$, 分别取鸡蛋数 $m = 10, ..., 100$, 统计运行时间.
- 以 $n = 1000, m = 50$ 为基准, 计算理论运行时间.

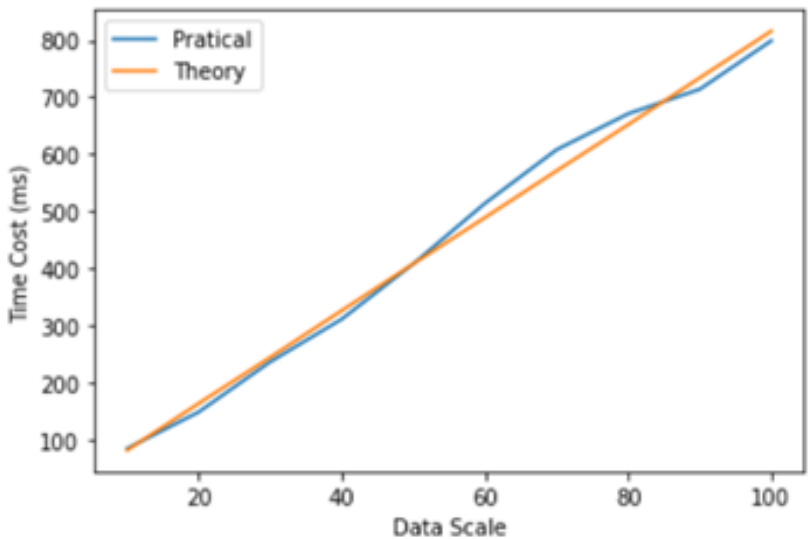

Figure 4: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

- 分析: 实际运行时间与鸡蛋数 m 成线性关系.

5. DP (优化 I: 二分最优转移点)

DP (优化 I: 二分最优转移点)

- 同 DP (朴素).
- 状态设计: $dp[i][j]$ 表示用 j 个鸡蛋找到 i 层的分界楼层 f 所需的最小实验次数.
- 初始条件: $dp[i][j] = \begin{cases} i, j = 1 \\ INF, 2 \le j \le m \end{cases} \quad (1 \le i \le n).$
- 最终答案: $ans = dp[n][m]$.

DP 的状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

- 对固定的 i 和 j, 注意到 k 从 1 增大到 i 时:

(1) $dp[k-1][j-1]$ 不减, 因为鸡蛋数 $(j-1)$ 固定时, 楼层越多, 所需的实验次数越多.

$k = 1$ 时, $dp[k-1][j-1]$ 取得最小值 $dp[0][j-1] = 0$.

$k = i$ 时, $dp[k-1][j-1]$ 取得最大值 $dp[i-1][j-1]$.

(2) $dp[i-k][j]$ 不增, 因为鸡蛋数 j 固定时, 楼层越少, 所需的实验次数越少.

$k = 1$ 时, $dp[i-k][j]$ 取得最大值 $dp[i-1][j]$.

$k = i$ 时, $dp[i-k][j]$ 取得最大值 $dp[0][j] = 0$.

DP 的状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

$\max\{dp[k-1][j-1], dp[i-k][j]\}$ 的图象如下图蓝色实线所示 (事实上函数的图象是离散的点):

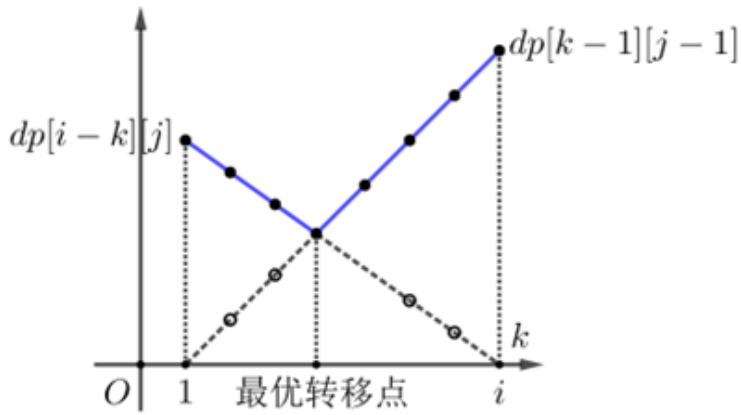

Figure 5: $\max\{dp[k-1][j-1], dp[i-k][j]\}$ 的图象

则最优转移点为两实线的交点. 因 $dp[k-1][j-1]$ 和 $dp[i-k][j]$ 的自变量和取值都是整数, 故交点必存在.

DP 的状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

实现时, 将 DP (朴素) 中的最内层循环改为二分最优转移点.

- 时空复杂度:

(1) 状态数 $O(nm)$, 转移 $O(\log n)$, 总时间复杂度 $O(nm \log n)$.

(2) 总空间复杂度 $O(nm)$.

- 小数据验证: 用 Leetcode 887 鸡蛋掉落验证正确性.

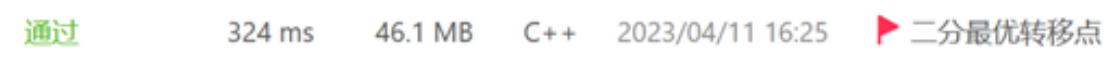

Figure 6: 二分最优转移点的正确性验证

DP (优化 I: 二分最优转移点) 的实际运行效率

- 运行效率与楼层数 n 的关系: 固定鸡蛋数 $m = 100$, 分别取楼层数 $n = 1000, ..., 1e5$, 统计运行时间.
- 以 $n = 5000, m = 100$ 为基准, 计算理论运行时间.

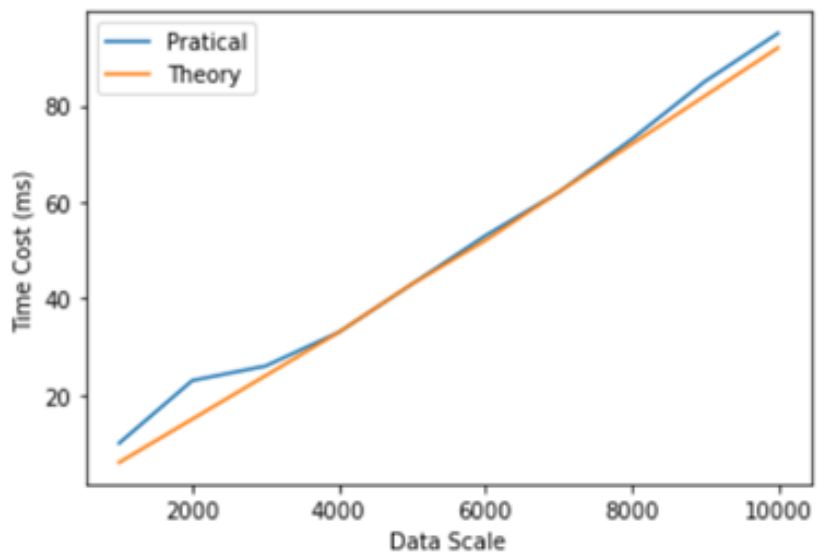

Figure 7: 实际运行时间与理论运行时间随楼层数 n 的变化关系

DP (优化 I: 二分最优转移点) 的实际运行效率

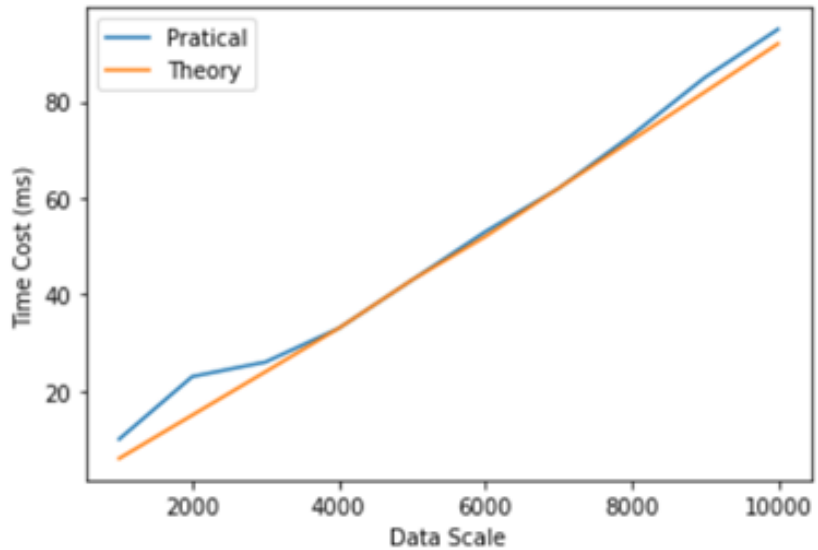

Figure 8: 实际运行时间与理论运行时间随楼层数 n 的变化关系

- 分析: 数据范围较小时, 实际运行时间略高于理论运行时间; 数据范围适中时, 实际运行时间与楼层数 n 成线性关系, 因 n 较小, $O(\log n)$ 对时间复杂度的贡献可忽略; 数据范围较大时, $O(\log n)$ 对时间复杂度的贡献不可忽略.

DP (优化 I: 二分最优转移点) 的实际运行效率

- 运行效率与鸡蛋数 m 的关系: 固定楼层数 $n = 1e4$, 分别取鸡蛋数 $m = 100, ..., 1000$, 统计运行时间.
- 以 $n = 1e4, m = 500$ 为基准, 计算理论运行时间.

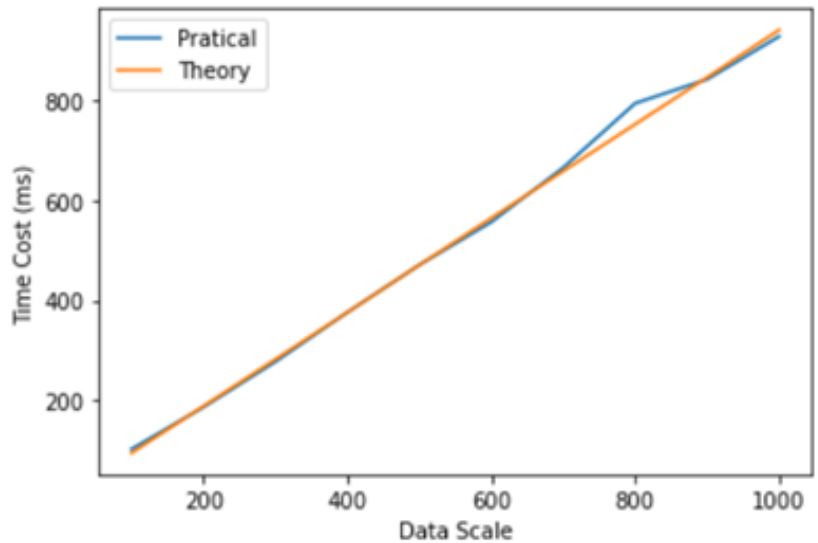

Figure 9: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

- 分析: 实际运行时间与鸡蛋数 m 成线性关系.

6. DP (优化 II: 决策单调性)

DP (优化 II: 决策单调性)

- 同 DP (朴素).
- 状态设计: $dp[i][j]$ 表示用 j 个鸡蛋找到 i 层的分界楼层 f 所需的最小实验次数.
- 初始条件: $dp[i][j] = \begin{cases} i, j = 1 \\ INF, 2 \le j \le m \end{cases} \quad (1 \le i \le n).$
- 最终答案: $ans = dp[n][m]$.

DP 的状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

- 对固定的 j 和 k, 随着 i 的增大, $dp[k-1][j-1]$ 不变, 但 $dp[i-k][j]$ 不减.
- 随着 k 的增大, $dp[k-1][j-1]$ 不减, $dp[i-k][j]$ 不增, 则随着 i 的增大, $dp[i-k][j]$ 的最大值不减.
- 在 $dp[k-1][j-1]$ 不变时, 如下图, 两线段的交点 (最优转移点) 不左移, 即决策有单调性.

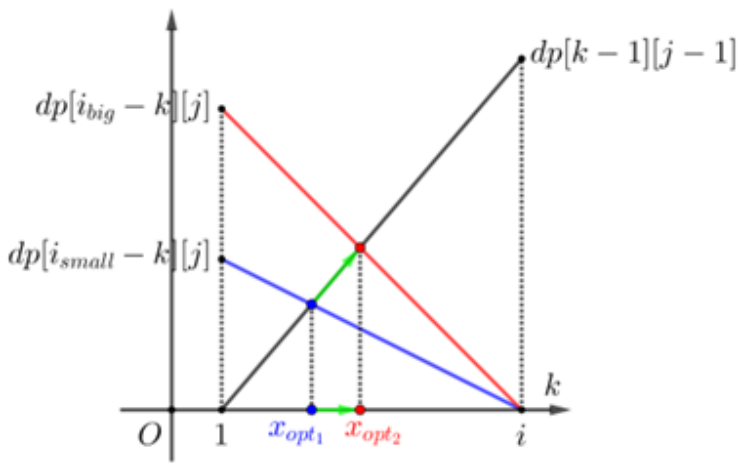

Figure 10: 决策单调性示意图

DP 的状态转移方程

$$dp[i][j] = \min_{1 \le k \le i} \{\max\{dp[k-1][j-1], dp[i-k][j]\} + 1\}.$$

实现时用变量记录最后一个最优转移点, 每次转移时尝试右移最优转移点.

- 时空复杂度:

(1) 状态数 $O(nm)$, 因最优转移点从 1 单调右移至 n, 故转移的均摊时间复杂度 $O(1)$, 总均摊时间复杂度 $O(nm)$.

(2) 总空间复杂度 $O(nm)$.

- 小数据验证: 用 Leetcode 887 鸡蛋掉落验证正确性.

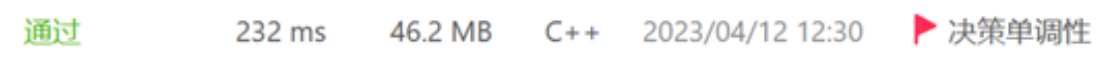

Figure 11: 决策单调性的正确性验证

DP (优化 II: 决策单调性) 的实际运行效率

- 运行效率与楼层数 n 的关系: 固定鸡蛋数 $m = 100$, 分别取楼层数 $n = 1000, ..., 1e5$, 统计运行时间.
- 以 $n = 5000, m = 100$ 为基准, 计算理论运行时间.

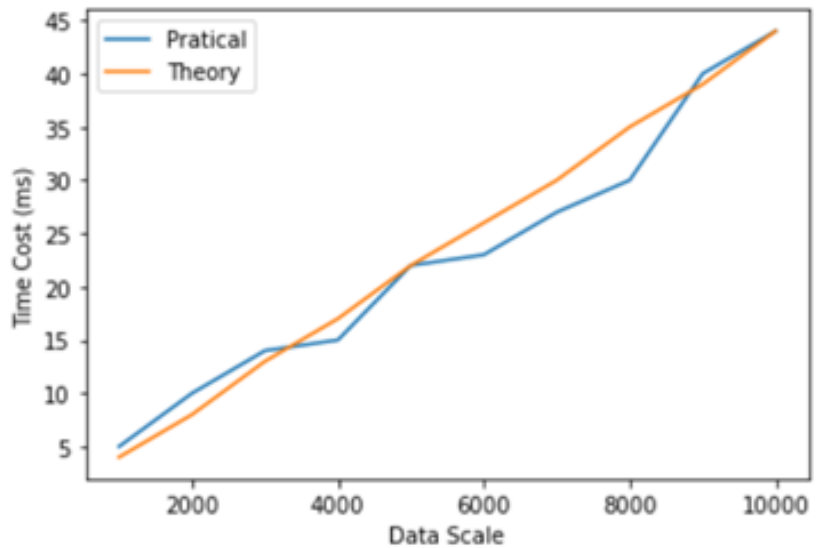

Figure 12: 实际运行时间与理论运行时间随楼层数 n 的变化关系

DP (优化 II: 决策单调性) 的实际运行效率

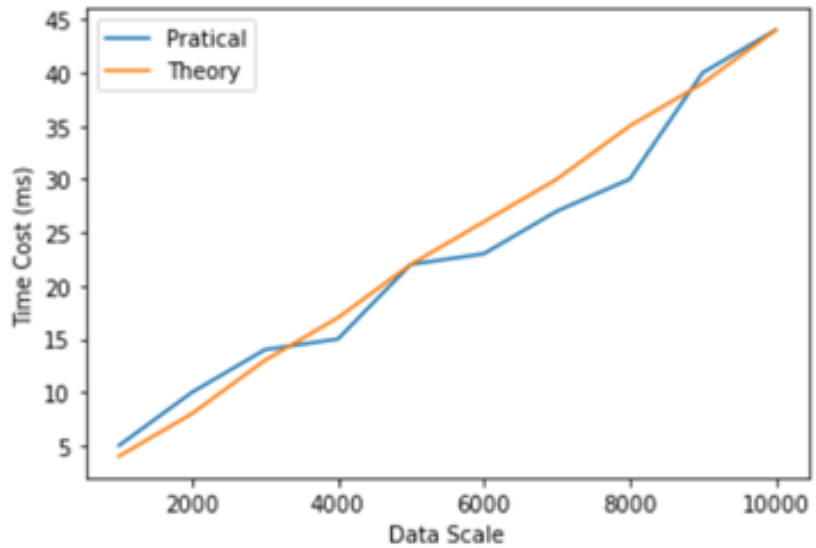

Figure 13: 实际运行时间与理论运行时间随楼层数 n 的变化关系

- 分析: 实际运行时间与楼层数 n 成线性关系. 上图中实际运行时间的曲线较为曲折, 但实际只与理论运行时间相差几毫秒, 是可接受的误差.

DP (优化 II: 决策单调性) 的实际运行效率

- 运行效率与鸡蛋数 m 的关系: 固定楼层数 $n = 1e4$, 分别取鸡蛋数 $m = 100, ..., 1000$, 统计运行时间.
- 以 $n = 1e4, m = 500$ 为基准, 计算理论运行时间.

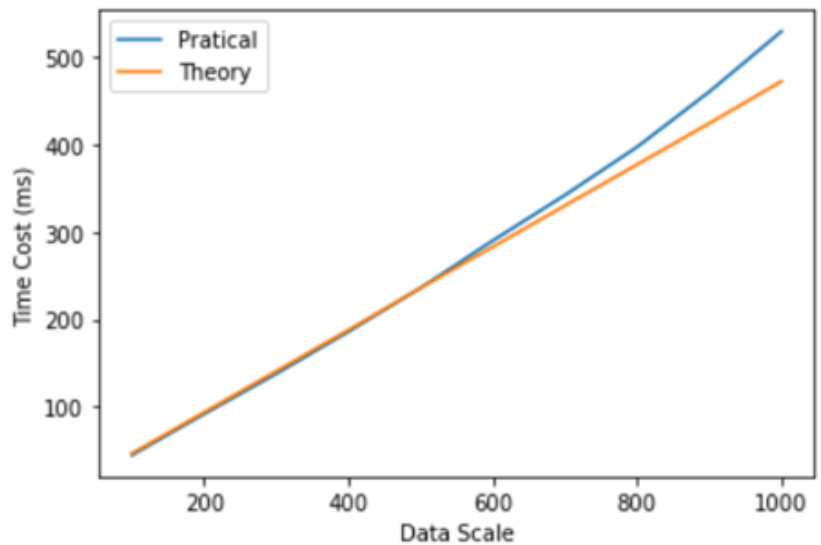

Figure 14: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

DP (优化 II: 决策单调性) 的实际运行效率

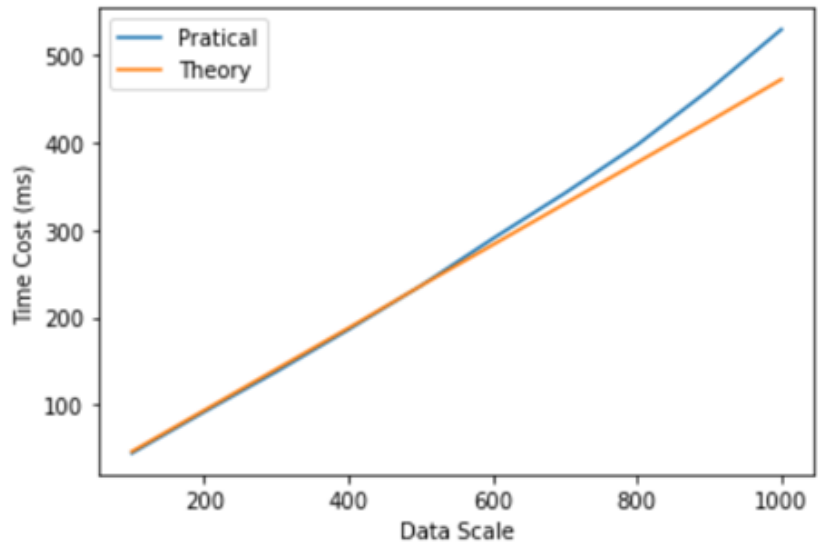

Figure 15: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

- 分析: 数据范围较小时, 实际运行时间与鸡蛋数 m 成线性关系; 数据范围较大时, 实际运行时间高于理论运行时间. 这表明: 本算法的时间复杂度只是均摊时间复杂度, 当 m 较大时, 可能稍有增大.

7. DP (优化 III: 状态设计优化)

DP (优化 III: 状态设计优化)

DP (朴素) 中 $dp[i][j]$ 表示用 j 个鸡蛋找到 i 层楼的分界楼层 f 所需的最小实验次数.

- 状态设计: 将 DP 的维度与表示的值交换, $dp[i][j]$ 表示用 i 个鸡蛋在 j 步内可确定分解楼层的楼层数的最大值.
- 初始条件: $dp[i][1] = 1 \ \ (1 \le i \le m)$.
- 最终答案: $s.t.\ dp[m][step] \ge n$ 的 $step$ 的最小值.

DP 的状态转移方程

- 按最后一个鸡蛋的情况分类:

(1) 若鸡蛋打破, 则可用剩下的 $(i–1)$ 个鸡蛋找到往下的 $(j–1)$ 层楼的分界楼层, 可确定分界楼层的楼层数的最大值为 $dp[i-1][j-1]$.

(2) 若鸡蛋未打破, 则可用剩下的 i 个鸡蛋找到往上的 $(j–1)$ 层的分界楼层, 可确定分界楼层的楼层数的最大值为 $dp[i][j-1]$.

故状态转移方程 $dp[i][j] = dp[i-1][j-1] + dp[i][j-1] + 1$.

DP 的状态转移方程

$$dp[i][j] = dp[i-1][j-1] + dp[i][j-1] + 1.$$

- 时空复杂度:

(1) 状态数 $O(nm)$, 转移 $O(1)$, 总时间复杂度 $O(nm)$. 但事实上, 该时间复杂度的上界非常宽松, 实际运行中几乎不会达到该上界. 查阅文献知, 更准确的时间复杂度为 $O\left(m\sqrt[m]{n}\right)$, 但实际运行效率更接近 $O(m\log n)$.

(2) 总空间复杂度 $O(nm)$.

- 小数据验证: 用 Leetcode 887 鸡蛋掉落验证正确性.

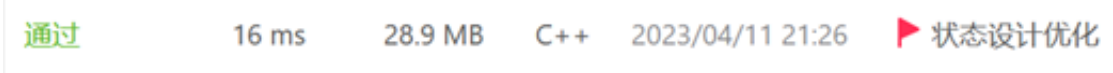

Figure 16: 状态设计优化的正确性验证

关于“内存连续访问时效率高”的讨论

(1) 先枚举 DP 的第二维, 再枚举 DP 的第一维, 代码如下图所示:

```cpp
// dp[i][j]表示用i个鸡蛋在j步内可确定分界楼层的楼层数的最大值
vector<vector<int>> dp(m + 1, vector<int>(n + 1));
for (int i = 1; i <= m; i++) dp[i][1] = 1;  // 初始条件

for (int j = 2; j <= n; j++) {  // 枚举步数
    for (int i = 1; i <= m; i++) {  // 枚举鸡蛋数
        dp[i][j] = dp[i - 1][j - 1] + dp[i][j - 1] + 1;
        if (dp[i][j] >= n) return j;
    }
}
return n;
```

Figure 17: 内存不连续访问的代码 I

此时内存不连续访问, 在 Leetcode 上的提交结果如下图所示:

Figure 18: 内存不连续访问的效率

关于“内存连续访问时效率高”的讨论

(2) 先枚举 DP 的第一维, 再枚举 DP 的第二维, 代码如下图所示:

```
// dp[i][j]表示用j个鸡蛋在i步内可确定分界楼层的楼层数的最大值
vector<vector<int>> dp(n + 1, vector<int>(m + 1));
for (int j = 1; j <= m; j++) dp[1][j] = 1;  // 初始条件

for (int i = 2; i <= n; i++) {  // 枚举步数
    for (int j = 1; j <= m; j++) {  // 枚举鸡蛋数
        dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j] + 1;
        if (dp[i][j] >= n) return i;
    }
}
return n;
```

Figure 19: 内存连续访问的代码 II

此时内存连续访问, 在 Leetcode 上的提交结果如下图所示:

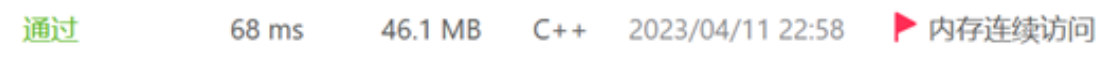

Figure 20: 内存连续访问的效率

关于“内存连续访问时效率高”的讨论

上述的实验结果与“内存连续访问时效率高”的经验不同, 可能的原因如下:

(1) 可能不成立的解释:

(i) **代码 I**的第一次转移为 $dp[1][2] = dp[0][1] + dp[1][1] + 1$. 计算等号右边时, 先取出 $dp[0][1]$ 和 $dp[1][1]$, 同时将 $dp[][]$ 的第 0、1 行放入 cache, 计算的结果赋值给 $dp[1][2]$, 发生 cache hit.

(ii) **代码 II**的第一次转移为 $dp[2][1] = dp[1][0] + dp[1][1] + 1$. 计算等号右边时, 先取出 $dp[1][0]$ 和 $dp[1][1]$, 同时将 $dp[][]$ 的第 1 行放入 cache, 计算的结果赋值给 $dp[2][1]$, 发生 cache miss.

关于“内存连续访问时效率高”的讨论

上述的实验结果与“内存连续访问时效率高”的经验不同, 可能的原因如下:

(2) 正确的解释:

(i) Leetcode 上本题的数据范围为 $m \leq 100, n \leq 1\text{e}5$. **代码 I**定义 $dp[][]$ 时调用了 $(m+1)$ 次 vector 的构造函数, **代码 II**定义 $dp[][]$ 时调用了 $(n+1)$ 次 vector 的构造函数. 上述的运行时间差异主要源于调用构造函数的开销, 因为分配内存的操作的时间复杂度可认为与使用次数成正比.

(ii) 为验证该结论, 进行了 $n = 30, m = 3\text{e}5$ 的实验, 发现内存连续访问的**代码 II**的效率高于内存不连续访问的**代码 I**, 故统计运行时间时不应统计定义 vector 的时间.

DP (优化 III: 状态设计优化) 的实际运行效率

- 下面统计效率时采用内存连续访问的**代码 II**.
- 在运行时间的测定上, 下面的数据范围可进一步过大, 但这面临着无法定义过大的数组的问题, 故下面的数据范围中, 对运行时间影响较大的鸡蛋数取值较大, 影响较小的楼层数 n 取值较小.

DP (优化 III: 状态设计优化) 的实际运行效率

- 运行效率与楼层数 n 的关系: 固定鸡蛋数 $m = 1e5$, 分别取楼层数 $n = 100, ..., 1000$, 统计运行时间.
- 以 $n = 500, m = 1e5$ 为基准, 计算理论运行时间.

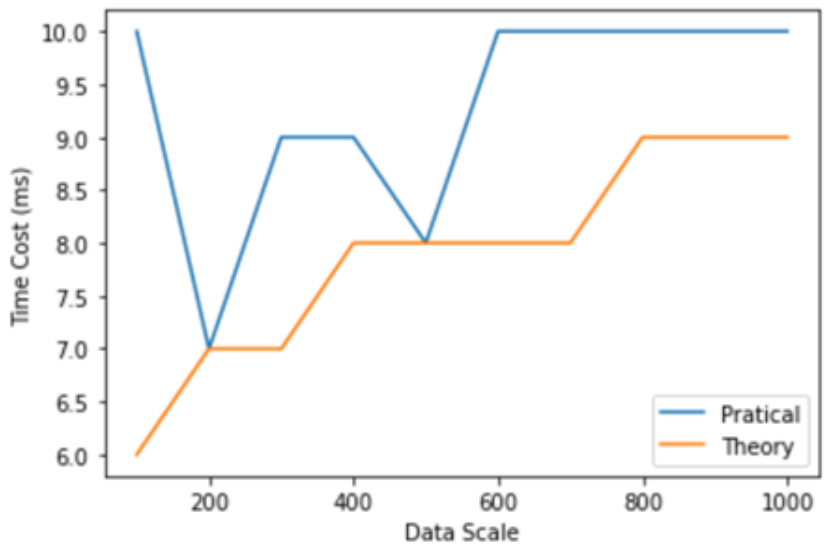

Figure 21: 实际运行时间与理论运行时间随楼层数 n 的变化关系

DP (优化 III: 状态设计优化) 的实际运行效率

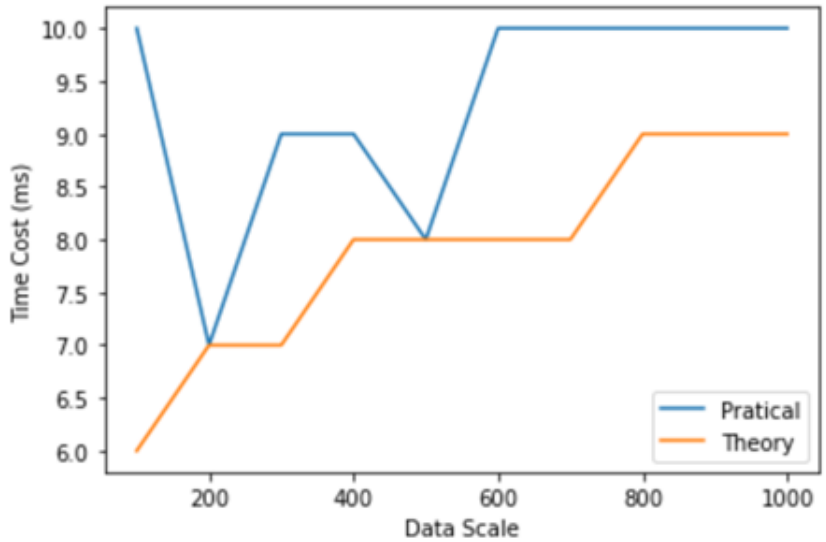

Figure 22: 实际运行时间与理论运行时间随楼层数 n 的变化关系

- 分析:

(1) 实际运行时间整体高于理论运行时间, 但整体增长趋势与理论运行实际按基本相同.

(2) 楼层数 n 对算法复杂度的影响较小.

DP (优化 III: 状态设计优化) 的实际运行效率

- 运行效率与鸡蛋数 m 的关系: 固定楼层数 $n = 1e4$, 分别取鸡蛋数 $m = 100, ..., 1000$, 统计运行时间.
- 以 $n = 1e4, m = 500$ 为基准, 计算理论运行时间.

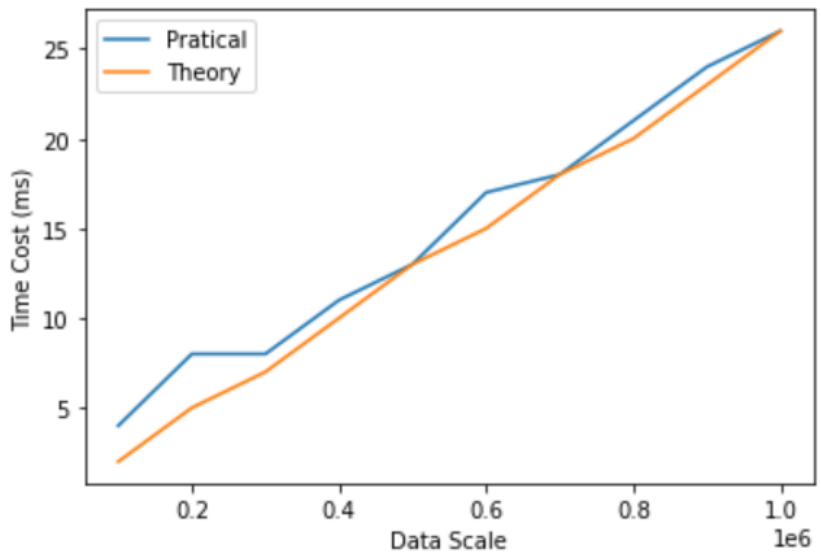

Figure 23: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

DP (优化 III: 状态设计优化) 的实际运行效率

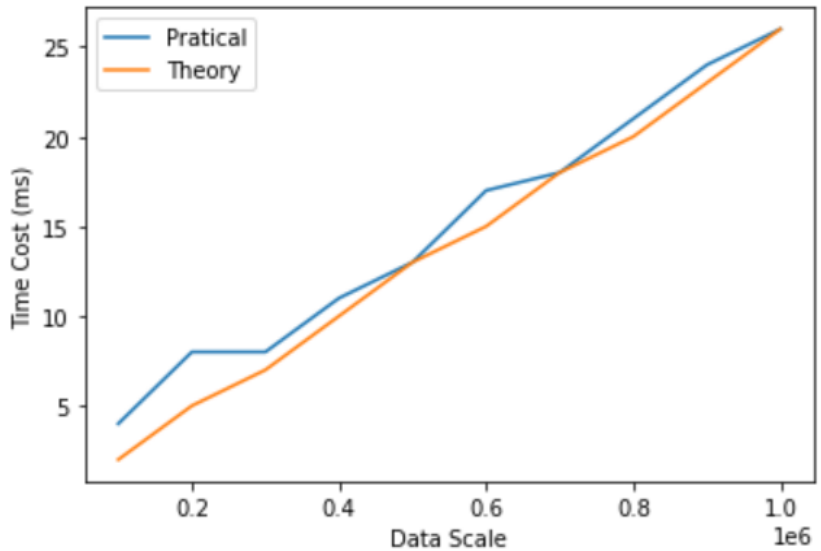

Figure 24: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

- 分析:

(1) 实际运行效率与鸡蛋数 m 基本成线性关系, 这与 n 较小时, $O(\log n)$ 或 $O\left(\sqrt[m]{n}\right)$ 都很接近 1 对应.

(2) 相比于楼层数 n, 鸡蛋数 m 对算法复杂度的影响较大.

8. DP (优化 IV: 滚动数组)

DP (优化 IV: 滚动数组)

- 同 DP (优化 III: 状态设计优化).
- 状态设计: $dp[i][j]$ 表示用 j 个鸡蛋找到 i 层的分界楼层 f 所需的最小实验次数.
- 初始条件: $dp[i][j] = \begin{cases} i, j = 1 \\ INF, 2 \le j \le m \end{cases} \quad (1 \le i \le n).$
- 最终答案: $ans = dp[n][m]$.
- 状态转移方程: $dp[i][j] = dp[i-1][j-1] + dp[i][j-1] + 1$.

DP (优化 IV: 滚动数组)

$$dp[i][j] = dp[i-1][j-1] + dp[i][j-1] + 1.$$

- 思路: 注意到 $dp[i][]$ 只与 $dp[i-1][]$ 有关, 故可用滚动数组优化.
- 实现时, 用两个长度为 $(m+1)$ 的 vector 滚动即可.
- 时空复杂度:

(1) 状态数 $O(nm)$, 转移 $O(1)$, 总时间复杂度 $O(nm)$. 同 DP (优化 III: 状态设计优化), 更准确的时间复杂度为 $O\left(m\sqrt[m]{n}\right)$, 但实际运行效率接近 $O(m\log n)$.

(2) 总空间复杂度 $O(2\cdot m) = O(m)$.

- 小数据验证: 用 Leetcode 887 鸡蛋掉落验证正确性.

Figure 25: 滚动数组优化的正确性验证

DP (优化 IV: 滚动数组)

- 因定义用于滚动的数组 (或使用全局数组时的清空) 也是算法运行的一部分, 故统计时间时应计入 DP 过程中调用构造函数的开销.
- 运行效率与楼层数 n 的关系: 固定鸡蛋数 $m = 1e7$, 分别取楼层数 $n = 100, ..., 1000$, 统计运行时间.
- 以 $n = 500, m = 1e7$ 为基准, 计算理论运行时间.

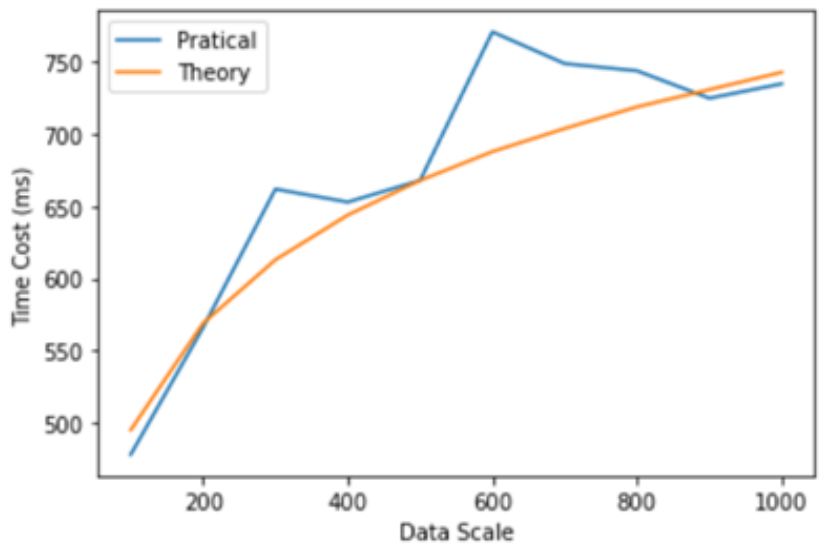

Figure 26: 实际运行时间与理论运行时间随楼层数 n 的变化关系

DP (优化 IV: 滚动数组) 的实际运行效率

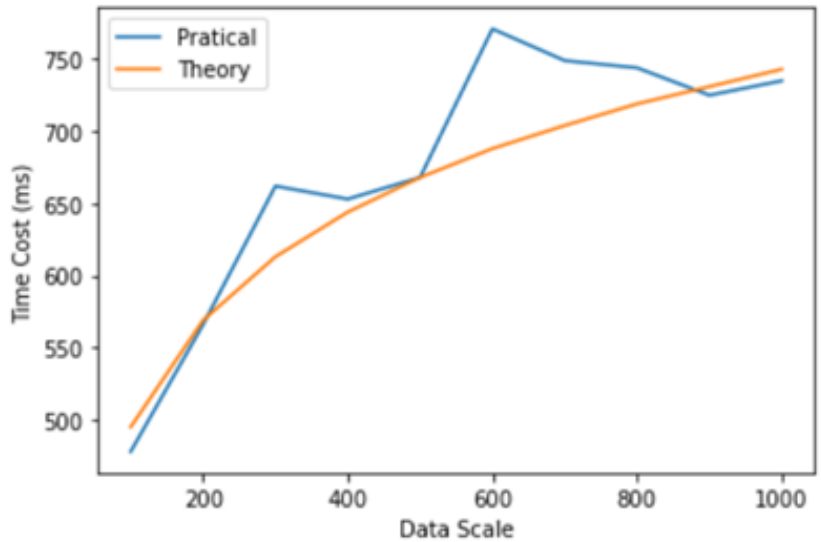

Figure 27: 实际运行时间与理论运行时间随楼层数 n 的变化关系

- 分析: 实际运行时间与楼层数 n 基本成 $O(\log n)$ 的增长, 但在某些数据, 如 $n = 300, 600, 700$ 时, 实际运行时间高于理论运行时间, 这可能与数据本身较特殊有关.

DP (优化 IV: 滚动数组) 的实际运行效率

- 运行效率与鸡蛋数 m 的关系: 固定鸡蛋数 $n = 1e4$, 分别取鸡蛋数 $m = 1e6, ..., 1e7$, 统计运行时间.
- 以 $n = 1e5, m = 5e6$ 为基准, 计算理论运行时间.

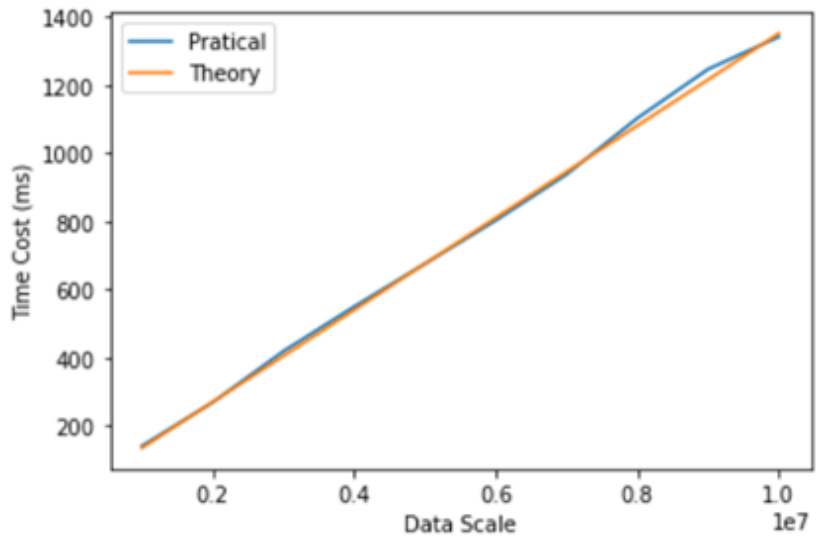

Figure 28: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

DP (优化 IV: 滚动数组) 的实际运行效率

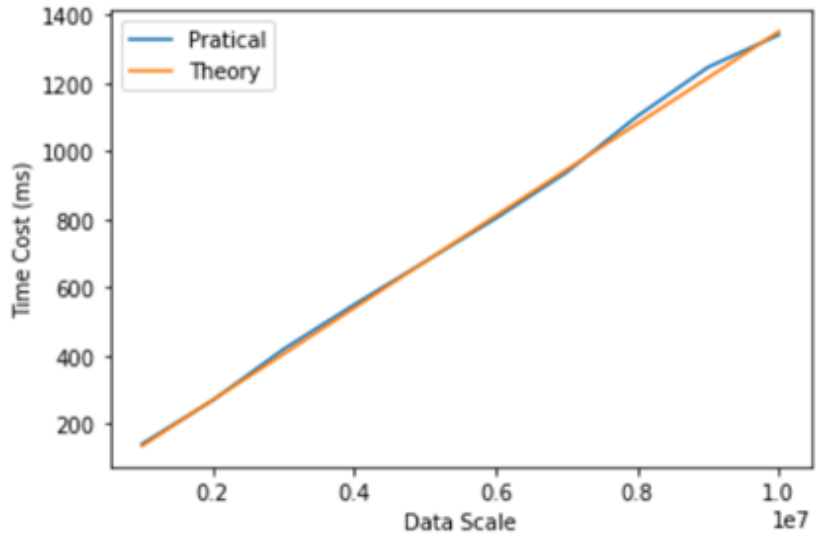

Figure 29: 实际运行时间与理论运行时间随鸡蛋数 m 的变化关系

- 分析: 实际运行效率与鸡蛋数 m 基本成线性关系, 这与 n 较小时, $O(\log n)$ 或 $O\left(\sqrt[m]{n}\right)$ 都很接近 1 对应.

有限时间内能处理的最大数据

- 分析: 根据时间复杂度和空间复杂度, 该算法可行性的瓶颈在于本地栈空间较小, 无法开更大的数组; 实际运行效率的瓶颈在于多次调用构造函数带来的开销. 前者可通过手动开大栈空间以定义更大的数组解决. 此外, 答案可作为实际运行效率的量度, 因为它一定程度上反映调用构造函数的次数.

有限时间内能处理的最大数据

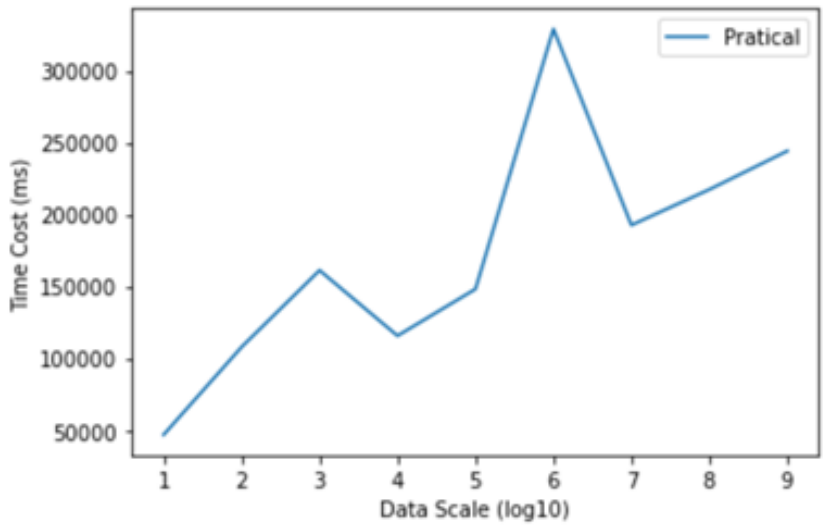

Figure 30: 实际运行时间随楼层数 n 的变化关系

- 分析:

(1) 实际运行时间不随楼层数 n 的增大而严格增大, $n = 1e4$ 和 $n = 1e5$ 时的实际运行时间小于 $n = 1000$ 时的实际运行时间, $n = 1e7$ 时的实际运行时间小于 $n = 1e6$ 时的实际运行时间.

有限时间内能处理的最大数据

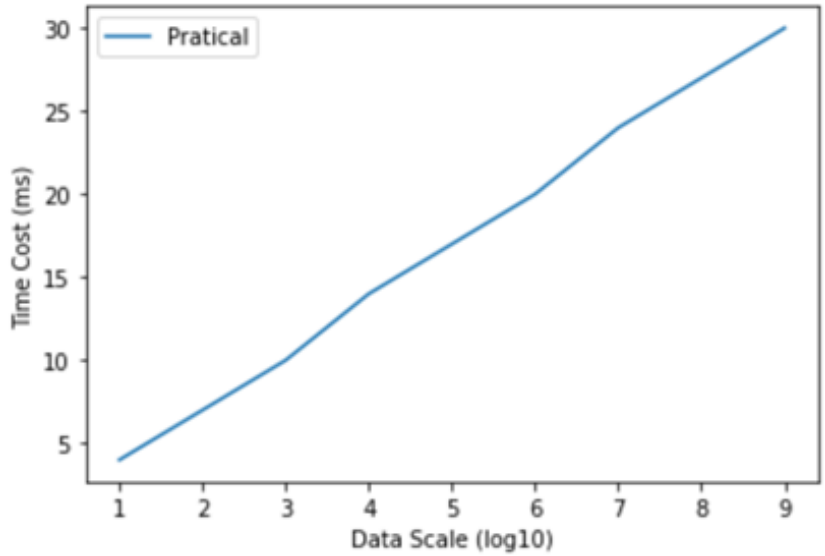

Figure 31: 答案数随楼层数 n 的变化关系

- 分析:

(2) 整体趋势上, 答案随楼层数 n 的增大而增大, 而答案一定程度上反映 DP 过程中调用构造函数的次数, 这表明: 进行多次分配空间的操作时, 编译器会进行优化, 以提高效率. 此外, 随着楼层数 n 的增大, DP 值增大的速度明显加快, 以至于在较短的时间内增大至 n, 进而跳出循环.

有限时间内能处理的最大数据

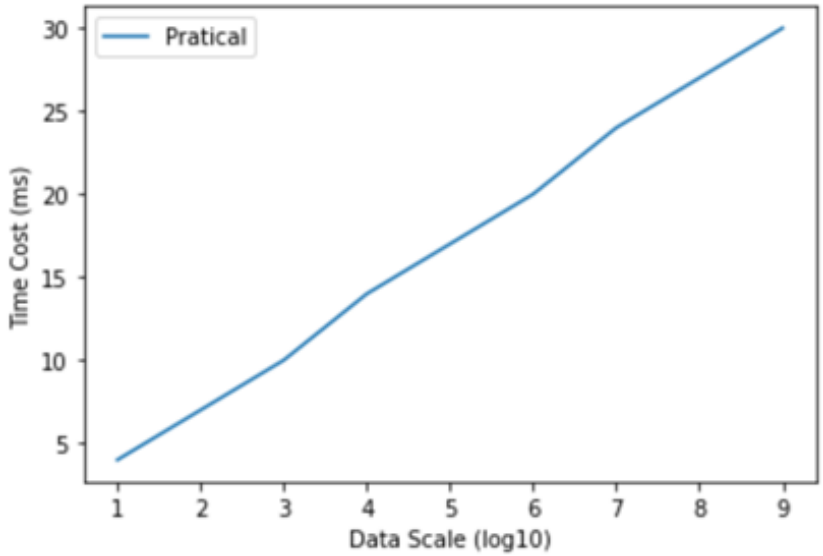

Figure 32: 答案数随楼层数 n 的变化关系

- 分析:

(3) 答案数与楼层数 n 成线性关系, 且楼层数 n 每增大 10 倍, 答案的增量在 $3 \sim 4$ 间, 约为 $\log_2 10$.

- 实践中, 在不借助外存的条件下, 有限时间内能处理的最大数据为 $n = 2\text{e}9, m = 1\text{e}9$, 耗时 332733 ms, 答案为 31.

谢　谢!

Thank you!