

深圳大学实验报告

课程名称： 算法设计与分析

实验名称： 动态规划—鸡蛋掉落问题

学院： 计算机与软件学院 专业： 计科

报告人： 欧阳宇杰 学号： 2021150143 班级： 计科2班

同组人： 欧阳宇杰

指导教师： 杜智华

实验时间： 2023年4月21日~2023年5月8日

实验报告提交时间： 2023年5月9日

教务处制

一、实验目的：

- (1) 掌握动态规划算法设计思想。
- (2) 掌握鸡蛋坠落问题的动态规划解法。

二、内容：

动态规划将问题划分为更小的子问题，通过子问题的最优解来重构原问题的最优解。动态规划中的子问题的最优解存储在一些数据结构中，这样我们就不必在再次需要时重新处理它们。任何重复调用相同输入的递归解决方案，我们都可以使用动态规划对其进行优化。

鸡蛋掉落问题是理解动态规划如何实现最佳解决方案的一个很好的例子。问题描述如下：



我们需要用鸡蛋确认在多大的楼层鸡蛋落下来会破碎，这个刚刚使鸡蛋破碎的楼层叫门槛层，门槛楼层是鸡蛋开始破碎的楼层，上面所有楼层的鸡蛋也都破了。另外，如果鸡蛋从门槛楼层以下的任何楼层掉落，它都不会破碎。如上图所示，如果有 5 层，我们只有 1 个鸡蛋，要找到门槛层，则必须尝试从每一层一层一层地放下鸡蛋，从第一层到最后一层，如果门槛层是第 k 层，那么鸡蛋就会在第 k 层抛下时破裂，应该做了 k 次试验。也就是说，如果有 k 层楼，1 个鸡蛋，最少的实验次数是 k 次。反之，如果有 e 个鸡蛋，楼层数是 0，则最少试验次数是 0，如果有 e 个鸡蛋，楼层数是 1，最少试验次数是 1。

如果有 6 层楼，三个鸡蛋，需要的最少试验次数是 3；如果有 5 层楼，三个鸡蛋，需要的最少试验次数也是 3。

注意：我们不能随机选择任何楼层，例如，如果我们选择 4 楼并放下鸡蛋并且它打破了，那么它不确定它是否也从 3 楼打破。因此，我们无法找到门槛层，因为鸡蛋一旦破碎，就无法再次使用。

给定建筑物的一定数量的楼层（比如 f 层）和一定数量的鸡蛋（比如 e 鸡蛋），找出阈值地板必须执行的最少的鸡蛋掉落试验的次数，注意，这里要求的是试验的次数，不是

鸡蛋的个数。还要记住的一件事是，我们寻找的是找到门槛层所需的最少掉落试验次数，而不是门槛层下限本身。

问题约束条件：

- 从跌落中幸存下来的鸡蛋可以再次使用。
- 破蛋必须丢弃。
- 摔碎对所有鸡蛋的影响都是一样的。
- 如果一个鸡蛋掉在地上摔碎了，那么它从高处掉下来也会摔碎。
- 如果一个鸡蛋在跌落中幸存下来，那么它在较短的跌落中也能完整保留下来。

三、实验要求

- 1、给出解决问题的动态规划方程；
- 2、随机产生 f ， e 的值，对小数据模型利用蛮力法测试算法的正确性；
- 3、随机产生 f ， e 的值，对不同数据规模测试算法效率，并与理论效率进行比对，请提供能处理的数据最大规模，注意要在有限时间内处理完；
- 4、该算法是否有效率提高的空间？包括空间效率和时间效率。

四、实验步骤与结果

❖ 探究该问题的状态转移方程：

题目要求的门槛层 k 是使鸡蛋破裂的最小层数，而我定义楼层 a 满足 $0 \leq a \leq n$ ，任何从高于 a 的楼层落下的鸡蛋都会碎，从 a 楼层或比它低的楼层落下的鸡蛋都不会破。

从题目对门槛层 k 的定义和我对楼层 a 的定义可知， $k=a+1$ ，即楼层 k 是楼层 a 的上一层，这意味着找到了楼层 a 就找到了门槛层 k 。

题目要求找到门槛层所需的最少掉落试验次数，那么就可转化为找到确切的楼层 a 的最少掉落试验次数。

以下我将“掉落试验”简写成“操作”。

假设当有 e 个鸡蛋， f 层楼的时候，要确定 a 确切的值的最小操作次数是 $ans = dp(e, f)$ 次，状态表示成 (e, f) 。当我们从 x 楼扔鸡蛋的时候，只会出现以下两种情况：

- a. 鸡蛋没碎：那么鸡蛋个数 e 不变，那么状态变为 $(e, f-x)$ ，因为 a 只可能在上方的 $n-x$ 层楼中。也就是说，我们把原问题缩小成了一个规模为 $(e, f-x)$ 的子问题，即 $ans1 = dp(e, f-x)$ 。

举例：2 个鸡蛋，4 层楼，如果从 2 楼扔下鸡蛋，鸡蛋没碎，那么鸡蛋接下来要在 3 楼甚至 4 楼再扔下试试的，所以这个时候，就演变成了 2 个鸡蛋 2 层楼的问题；

- b. 鸡蛋碎了：鸡蛋个数-1，那么状态变为 $(e-1, x-1)$ ，因为 a 只可能在下方的 $x-1$ 层楼中了。也就是说，我们把原问题缩小成了一个规模为 $(e-1, x-1)$ 的子问题，即 $ans2 = dp(e-1, x-1)$ 。
- 举例：2 个鸡蛋，4 层楼，如果从 2 楼扔下鸡蛋，鸡蛋碎了，那么鸡蛋接下来要在 1 楼扔下试试的，所以这个时候，就演变成了 1 个鸡蛋 1 层楼的问题；

不管从 x 楼扔下的鸡蛋碎没碎，我们的操作次数一定要能得出确切的 a ，由于我们不知道 a 值，因此我们必须保证无论 a 的值如何我们都能找到确切的 a 值，那么我们就保证鸡蛋碎了的情况和鸡蛋没碎的情况都能找到确切的 a 值。所以对于某一个 x ，需要的操作次数为 $1 + \max(ans1, ans2)$ ，即鸡蛋碎了之后接下来需要的操作次数和鸡蛋没碎之后接下来需要的操作次数二者的最大值。又 ans 是 e 个鸡蛋， f 层楼的时候，要确定 a 确切的值的最小操作次数，这意味着 x 可以取 $1 \sim f$ ，而 ans 是 x 遍历所有情况得到的操作次数的最小值。

故我们可以列出状态转移方程如下：

$$dp(k, n) = 1 + \min(\max(dp(e, f - x), dp(e - 1, x - 1))) \quad 1 \leq x \leq f$$

❖ 根据状态转移方程写出蛮力法代码并尝试不断优化

根据上面的状态转移方程可以直接得到下面的第一版蛮力法代码：

```
1  int dp(int e, int f) {
2      if (f == 0 || f == 1 || e == 1) {
3          return f;
4      } //遇到特殊情况可以直接返回结果
5
6      int minimun = f; //最小操作次数先初始化为最大(楼层总数)
7      for (int x = 1; x <= f; x++) { //x从1遍历到f
8          minimun = min(minimun, 1 + max(dp(e - 1, x - 1), dp(e, f - x)));
9          //由状态转移方程导出的递推公式
10     }
11     return minimun;
12 }
13 int superEggDrop1(int e, int f) {
14     return dp(e, f);
15 }
```

上面的代码重复计算了很多节点。为了加快这个计算过程，一个简单的提升方法就是用空间换时间，把计算过的节点结果用二维数组储存起来，后面再用就直接查表（即存储结果的二维数组）。

根据上面的分析可写出下面的第二版代码（基础动态规划版）：

```
1  int superEggDrop2(int e, int f) {
2      int midRes[e+1][f+1]; //用于存储计算过的节点结果的二维数组
3
4      这一段利用边界条件初始化储存中间结果的二维数组
5
6      for (int k = 2; k <= e; k++) {
7          for (int n = 1; n <= f; n++) {
8              int tMinDrop = f;
9              for (int x = 1; x <= n; x++) {
10                 tMinDrop = min(tMinDrop, 1 + max(midRes[k - 1][x - 1], midRes[k][n - x]));
11             }
12             midRes[k][n] = tMinDrop;
13         }
14     }
15     return midRes[e][f];
16 }
```

这个解法利用了一个二维数组存储了计算过的结果（空间复杂度 $O(ef)$ ），使得时间复杂度降低到了 $O(ef^2)$ 。但是依然是一个三次方级别的时间复杂度，不够快，仍然有时间效率提高的空间，尝试继续优化。

我们观察到 $dp[e][f]$ 是一个关于 f 的单调递增函数，也就是在鸡蛋数 e 固定的情况下，楼层数 f 越多，需要的操作次数不会减少而是会增加。在上面的状态转移方程中第一项 $T_1(x) = dp(e, f - x)$ 是一个随着 x 的增加而单调递减的函数，第二项 $T_2(x) = dp(e - 1, x - 1)$ 是一个随着 x 的增加而单调递增的函数。我们要找到一个位置 x 使得 $T_1(x)$ 和 $T_2(x)$ 的最大值最小。

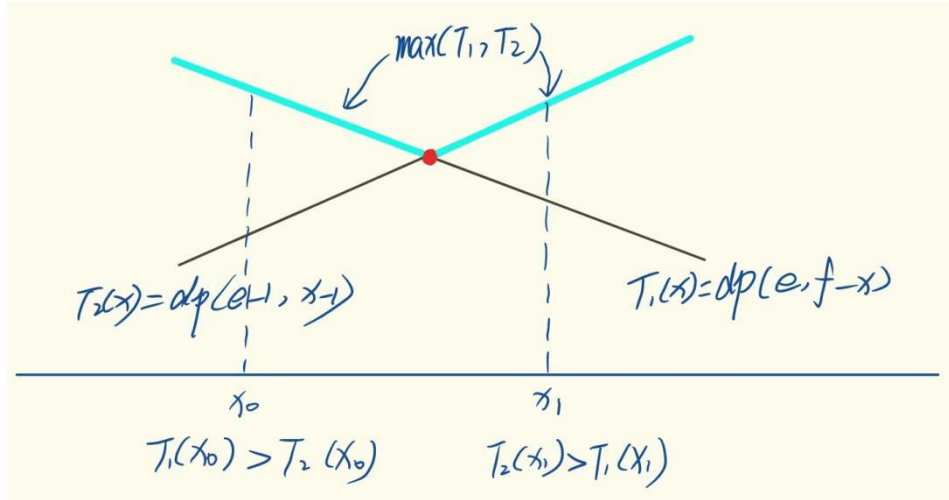


图 1: $T_1(x)$ 、 $T_2(x)$ 曲线和 $\max(T_1, T_2)$ 曲线及 x_0 与 x_1

如上图所示，绿色线即是 $T_1(x)$ 和 $T_2(x)$ 的最大值的变化曲线。如果 $T_1(x)$ 和 $T_2(x)$ 都是连续的函数，那么我们只需要找到这两个函数的交点（即图中的红点），在交点处就能满足这两个函数的最大值最小。但在本题中， $T_1(x)$ 和 $T_2(x)$ 是离散的， x 只能取 1, 2, 3 等等。在这种情况下，除非交点是大于 0 的正整数，否则 x 就取不到交点处。对于交点不为整数的情况，我们就要找到离交点最近的两个整数点，它们一个在交点左侧，另一个在交点右侧。假设左侧的为 x_0 ，右侧的为 x_1 ，我们只需要比较在 x_0 处和 x_1 处 $T_1(x)$ 和 $T_2(x)$ 的最大值，使得 $T_1(x)$ 和 $T_2(x)$ 的最大值较小的那个 x 就是我们在找的使得 $T_1(x)$ 和 $T_2(x)$ 的最大值最小的位置 x 。因为交点不是整数，且 x_0 和 x_1 是交点左右两侧离交点最近的两个整数点，那么容易得到 x_0 和 x_1 相差 1。

故我们可以用二分查找来寻找交点或 x_0 和 x_1 。

据上面的图 2 我们可以知道如果二分查找的 mid 位置有 $T_1 > T_2$ ，那么交点在 mid 右边，故在 mid 的右边继续找，如果二分查找的 mid 位置有 $T_1 < T_2$ ，那么交点在 mid 左边，所以在 mid 的左边继续找，直到 $low+1==high$ 或找到交点。

若是因为 $low+1==high$ 而跳出 $while$ 循环，则 low 就是 x_0 ， $high$ 就是 x_1 。那就只需要比较在 x_0 处和 x_1 处 T_1 和 T_2 的最大值，使得 T_1 和 T_2 的最大值较小的那个 x 就是我们在一直在找的使得 T_1 和 T_2 的最大值最小的位置 x 。

第三版代码如下：

```
1. unordered_map<int, int> memo; // 用 map 存储计算过的结果
2. int dp2(int e, int f) {
3.     if (该(e, f) 数对没有被计算过) {
4.         int ans; // 存储答案
5.         if (楼层数为 0) {
6.             ans = 0;
7.         }
8.         else if (鸡蛋数为 1) {
9.             ans = f;
```

```

10.         }
11.         else {
12.             int low = 1, high = f;
13.             while (low + 1 < high) {
14.                 int x = (low + high) / 2;
15.                 int t1 = dp2(e, f - x); //代表 T1 函数的值
16.                 int t2 = dp2(e - 1, x - 1); //代表 T2 函数的值
17.
18.                 if (t1 > t2) {
19.                     low = x; //交点在右边, 往右搜索
20.                 }
21.                 else if (t1 < t2) {
22.                     high = x; //交点在左边, 往左搜索
23.                 }
24.                 else //进入这表明找到了交点, 仅在交点为整数时会进
25.                     low = high = x;
26.             }
27.             //若是因为 low+1==high 而跳出 while 循环, 那么 low 就是 x0, high 就是 x1。
28.             ans = 1 + min(max(dp2(e - 1, low - 1), dp2(e, f - low)),
29.                            max(dp2(e - 1, high - 1), dp2(e, f - high)));
30.             memo[f * 10000 + e] = ans;
31.         }
32.         return memo[f * 10000 + e];
33.     }
34.     int superEggDrop3(int e, int f) {
35.         return dp2(e, f);
36.     }

```

时间复杂度：对于一个 (e, f) 数对，需要 $\log f$ 的时间进行二分查找，共有 $e \cdot f$ 个 (e, f) 数对，所以时间复杂度为 $O(ef \log f)$ 。

空间复杂度：对于一个 (e, f) 数对，需要一个空间来存储解，共有 $e \cdot f$ 个 (e, f) 数对，故空间复杂度为 $O(ef)$

❖ 尝试新思路，改变 dp 数组含义及状态转移方程以期优化时间与空间效率

那么还有更快的算法吗？该问题是否还有效率提高的空间？我们可以尝试一下逆向思维。

原来的 $dp[e][f]$ 数组的含义为：当有 e 个鸡蛋， f 层楼的时候，要确定 a 确切的值的最小操作次数。

我们可以改变 dp 数组的含义。我重新定义 $dp[e][m]$ 为：当有 e 个鸡蛋， m 次操作次数的情况下，能确定 a 的确切的值的最大楼层数。

那我们就将原问题转化为找出满足 $dp[e][m] \geq f$ 的最小的 m 。

那我们该怎么求 $dp[e][m]$ 呢？

若我们某层扔出一颗鸡蛋，就有两种情况

➤ 鸡蛋没碎： $dp[e][m-1]$ ，代表了在该楼层往上能确定 a 的确切的值的最大楼层数。

➤ 鸡蛋碎了： $dp[e-1][m-1]$ ，代表了在该楼层往下能确定 a 的确切的值的最大楼层数。
那么我们就可以写出以下的状态转移方程：

$$dp[e][m] = 1 + dp[e][m-1] + dp[e-1][m-1]。$$

下面我给出一个求 $dp[2][3]$ 的示例：

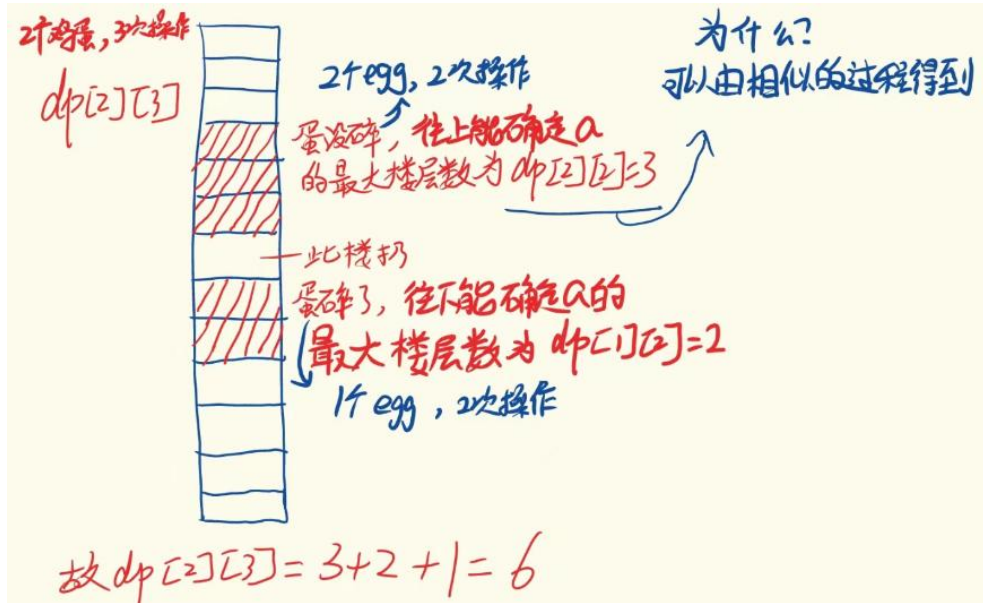


图 2：求 $dp[2][3]$ 的示例

m 的最大值是多少？由于楼层数 f 会给出，所以我们容易得到 m 最大只可能达到 f 。即 $m \leq f$ 。
故 dp 数组的第二个维度最大为 f 。

另外给出边界条件，用于初始化 dp 数组：

1. 对于 $m \geq 1$ ， $dp[1][m] = m$ ，含义：只有一个鸡蛋的情况下，有多少次操作就能确定多少层楼。
2. 对于 $e \geq 1$ ， $dp[e][1] = 1$ ，含义：若只能进行一次操作，那么无论多少个鸡蛋都只能确定 1 层楼。

根据上面的状态转移方程及其他分析可以轻易写出以下的第四版代码：

```
1. int superEggDrop4(int e, int f) {
2.     if (f == 1) return 1; // 特殊情况直接返回
3.     int dp[e+1][f+1];
4.     // 边界条件用于初始化
5.     for (m = 1; m <= f; m++)
6.         dp[1][m] = m;
7.     for (i = 1; i <= e; i++)
8.         dp[i][1] = 1;
9.     ans = -1;
10.    for (m = 2; m <= f; m++) { // m 代表操作次数, 从小到大遍历
11.        for (eNum = 1; eNum <= e; eNum++) {
12.            dp[eNum][m] = 1 + dp[eNum][m-1] + dp[eNum-1][m-1];
13.        }
14.        if (dp[e][m] >= f) {
```

```

15.         ans = m; //要求满足  $dp[e][m] \geq f$  的最小的  $m$ , 所以一旦满足就跳出
16.         break;
17.     }
18. }
19. return ans;
20. }

```

从上面的代码中容易知道该算法的时间复杂度为 $O(ef)$ 。空间复杂度也为 $O(ef)$ 。

现在我根据实验要求对小数据模型利用蛮力法验证第四版代码算法的正确性：

我利用随机数生成器随机产生了十组 (e, f) 数对来测试，鸡蛋数和楼层数都限制在 21 以内。结果展示如下：

鸡蛋数：20 楼层数：3	鸡蛋数：12 楼层数：20
蛮力法得答案为2	蛮力法得答案为5
第四版代码算法得答案为2	第四版代码算法得答案为5
鸡蛋数：19 楼层数：12	鸡蛋数：7 楼层数：20
蛮力法得答案为4	蛮力法得答案为5
第四版代码算法得答案为4	第四版代码算法得答案为5
鸡蛋数：21 楼层数：12	鸡蛋数：6 楼层数：11
蛮力法得答案为4	蛮力法得答案为4
第四版代码算法得答案为4	第四版代码算法得答案为4
鸡蛋数：17 楼层数：14	鸡蛋数：16 楼层数：20
蛮力法得答案为4	蛮力法得答案为5
第四版代码算法得答案为4	第四版代码算法得答案为5
鸡蛋数：18 楼层数：3	鸡蛋数：19 楼层数：18
蛮力法得答案为2	蛮力法得答案为5
第四版代码算法得答案为2	第四版代码算法得答案为5

图 3：蛮力法和第四版代码算法得到的答案对比

从上图对小规模模型的测试结果来说，对于同一组数据，蛮力法和第四版代码算法得出的答案均相同，这验证了第四版代码算法的正确性。

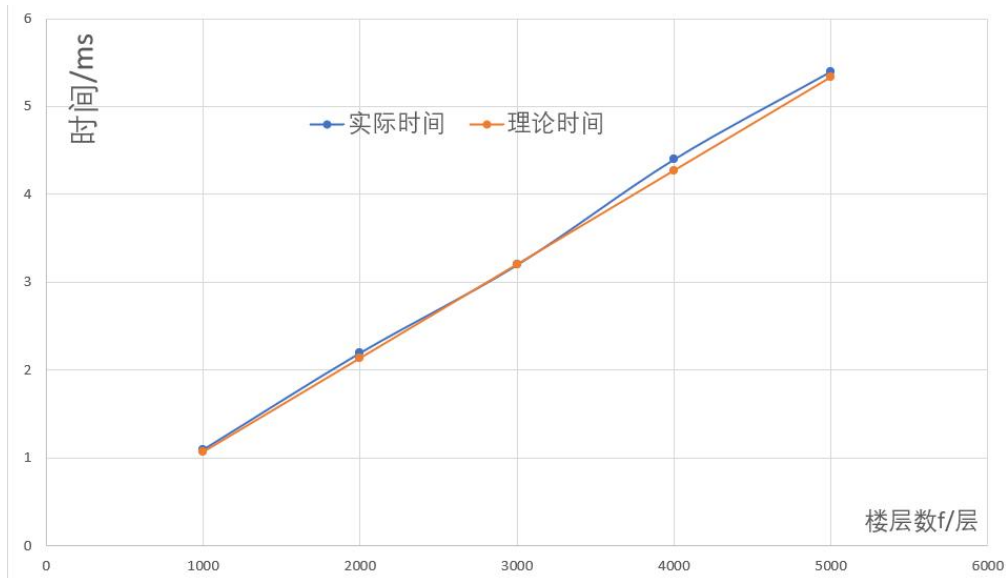
❖ 根据实验要求，我用不同数据规模测试第四版代码算法效率，并与理论效率进行比对

➤ 对比第四版代码算法的实际效率与理论效率(时间和空间复杂度都为 $O(ef)$)

- 1) 我固定 e 为 500, f 由 1000 变化到 5000, 实际运行时间取运行 10 次的平均值。我以 $f=3000$ 为基准点，根据其时间复杂度 $O(ef)$ 通过 $\frac{ef_1}{ef_2} = \frac{time1}{time2}$ 的比例关系，计算得到算法执行时间的理论值，通过与实际值的对比得到如下表，其中相差比例为（实际值-理论值）/实际值。

	e=500				
f/层	1000	2000	3000	4000	5000
实际时间/ms	1.1	2.2	3.2	4.3	5.4
理论时间/ms	1.067	2.133	3.2	4.267	5.33
相差比例	3%	3.05%	0%	0.77%	1.30%

根据上面的表格可绘制出下面的图象：

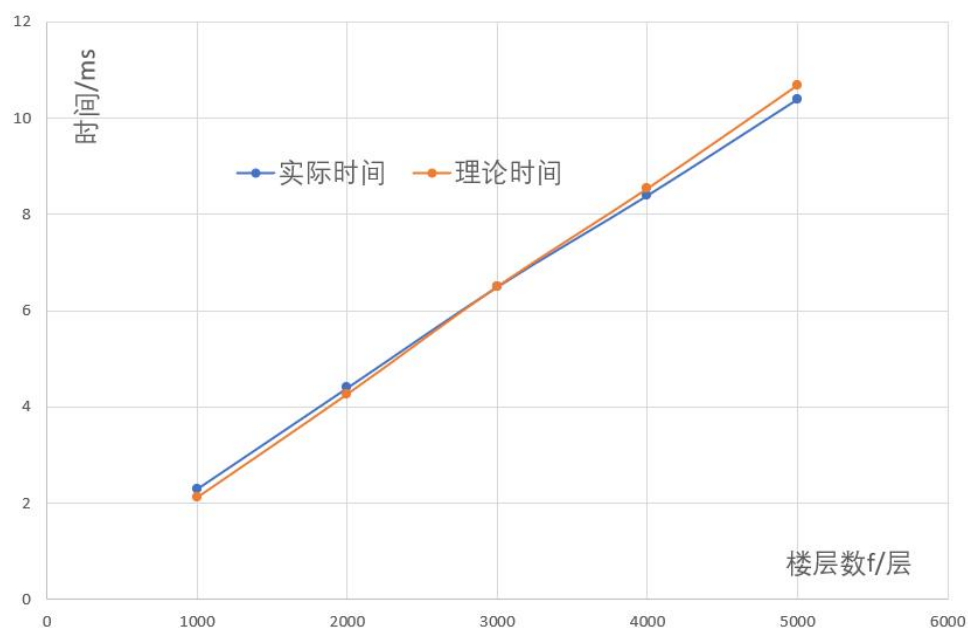


可以看到实际时间和理论时间基本相同，实际时间曲线基本拟合理论时间曲线。

- 2) 我固定 e 为 1000，f 由 1000 变化到 5000，实际运行时间取运行 10 次的平均值。我仍然以 f=3000 为基准点，根据其时间复杂度 $O(e f)$ 通过 $\frac{ef_1}{ef_2} = \frac{time_1}{time_2}$ 的比例关系，计算得到算法执行时间的理论值，通过与实际值的对比得到如下表，其中相差比例为（实际值-理论值）/实际值。

	e=1000				
f/层	1000	2000	3000	4000	5000
实际时间/ms	2.3	4.4	6.5	8.4	10.4
理论时间/ms	2.13	4.27	6.5	8.53	10.67
相差比例	7.39%	2.95%	0%	-1.55%	-2.60%

根据上面的表格可绘制出下面的图象：

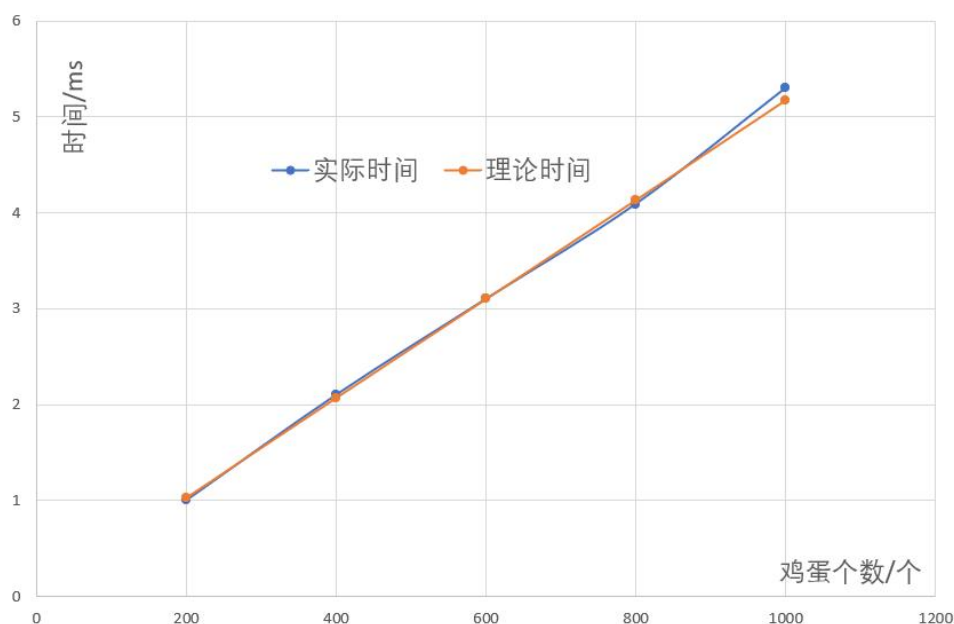


可以看到实际时间和理论时间基本相同，实际时间曲线基本拟合理论时间曲线。

- 3) 我固定 f 为 2000, f 由 200 变化到 1000, 实际运行时间取运行 10 次的平均值。我以 $e=600$ 为基准点, 根据其时间复杂度 $O(e f)$ 通过 $\frac{e_1 f}{e_2 f} = \frac{time1}{time2}$ 的比例关系, 计算得到算法执行时间的理论值, 通过与实际值的对比得到如下表, 其中相差比例为 (实际值-理论值)/实际值。

	f=2000				
e/个	200	400	600	800	1000
实际时间/ms	1	2.1	3.1	4.09	5.3
理论时间/ms	1.03	2.07	3.1	4.13	5.17
相差比例	-3%	1.43%	0%	-0.98%	2.45%

根据上面的表格可绘制出下面的图象:

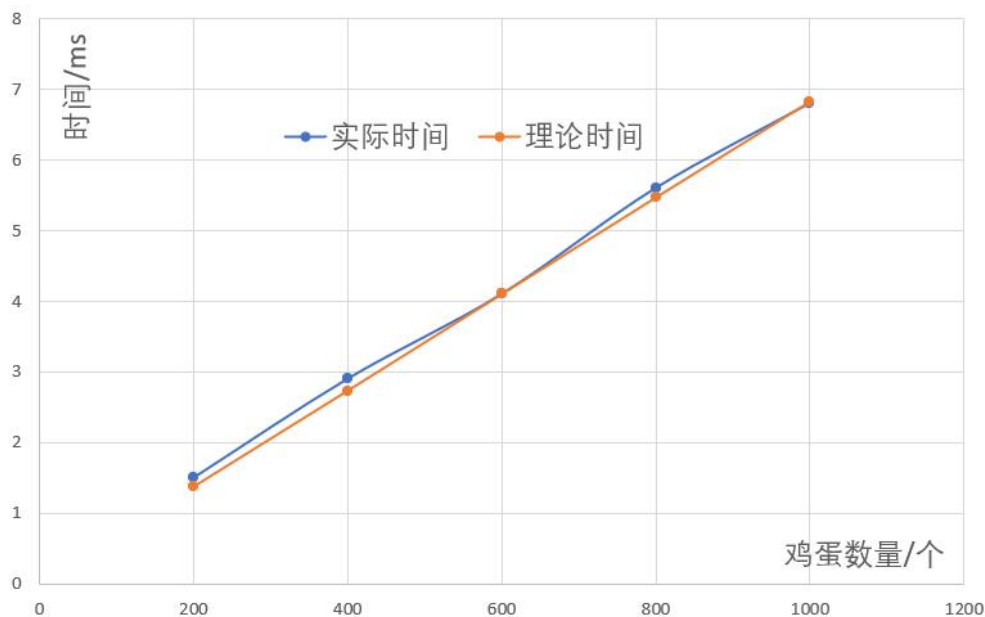


可以看到实际时间和理论时间基本相同，实际时间曲线基本拟合理论时间曲线。

- 4) 我固定 f 为 3000, f 由 200 变化到 1000, 实际运行时间取运行 10 次的平均值。我仍然以 $e=600$ 为基准点, 根据其时间复杂度 $O(e f)$ 通过 $\frac{e_1 f}{e_2 f} = \frac{time1}{time2}$ 的比例关系, 计算得到算法执行时间的理论值, 通过与实际值的对比得到如下表, 其中相差比例为 (实际值-理论值)/实际值。

	f=3000				
e/个	200	400	600	800	1000
实际时间/ms	1.5	2.9	4.1	5.6	6.8
理论时间/ms	1.37	2.73	4.1	5.47	6.83
相差比例	8.67%	5.86%	0%	2.32%	-0.44%

根据上面的表格可绘制出下面的图象:



可以看到实际时间和理论时间基本相同，实际时间曲线基本拟合理论时间曲线。

➤ 实验要求还要求提供能在有限时间内处理完的最大规模数据

我第四版代码算法的时间复杂度为 $O(e \cdot f)$ ，经验证即使是很大的数据规模也能在有限时间内处理完，这样制约最大数据规模的因素就变为了内存大小。我第四版代码算法的空间复杂度也为 $O(e \cdot f)$ 。只要内存够大不爆内存，我的算法都能在有限时间内处理完。

我假设可供使用的内存大小为 4GB，则其共可存放 $4 \cdot 1024 \cdot 1024 \cdot 1024 / 4 = 1073741824$ 个 int 数据。为 10^9 级别。故我假设 e 为 10^4 ， f 为 10^5 ，所需时间展示如下：

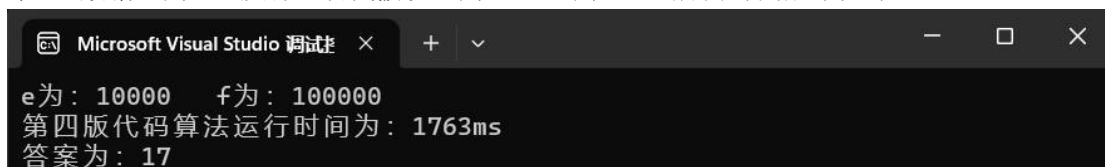


图 4: e 为 10^4 ， f 为 10^5 的情况下第四版代码所需时间

故我给出的结论为：能在有限时间内处理完的最大规模数据取决于内存，只要内存能存放得下数据，我的算法都能在有限时间内处理完。

❖ 尝试进一步优化算法

上面的算法的空间复杂度仍然有优化的空间，我们深入分析状态转移方程：

$$dp[e][m] = 1 + dp[e][m-1] + dp[e-1][m-1]$$

我们可以发现第 m 次操作结果只和第 $m-1$ 次操作结果相关，也就是第二个维度的参数 m 仅仅与 $m-1$ 有关，那么我们就可以将第二个维度删去，变为： $dp[e] = 1 + dp[e] + dp[e-1]$ ，这样在赋值前 dp 数组存储的是第 $m-1$ 次操作结果的数据，赋值后存储的是第 m 次操作结果的数据。

根据这样的思想我们可以写出下面的第五版代码：

```

1.  int superEggDrop5(int e, int f) {
2.      if (f == 1) return 1;
3.      vector<int> dp(e + 1, 0); // 仅开辟一维数组
4.      for (int i = 1; i <= e; i++)
5.          dp[i] = 1; // 边界条件用于初始化
6.      int ans = -1;
  
```

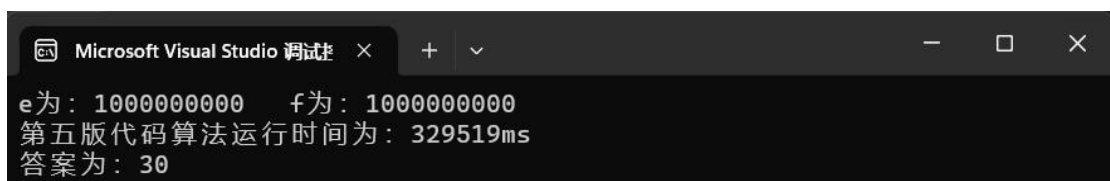
```

7.         for (int m = 2; m <= f; m++) { //m 代表操作次数
8.             //鸡蛋数量倒着遍历为了防止覆盖
9.             for (int eNum = e; eNum >= 1; eNum--) {
10.                dp[eNum] = 1 + dp[eNum] + dp[eNum - 1];
11.            }
12.            if (dp[e] >= f) {
13.                ans = m;
14.                break;
15.            }
16.        }
17.        return ans;
18.    }

```

这版的代码就是我最终算法(第五版)的代码，时间复杂度为 $O(ef)$ ，空间复杂度为 $O(e)$ 。

这最终的第五版代码算法时间复杂度与之前的第四版代码算法相同，故不再重复进行算法时间效率分析，但由于空间复杂度得到了优化由 $O(ef)$ 变为了 $O(e)$ ，所以其能在有限时间内处理完的最大规模数据改变了。我仍然假设可供使用的内存大小为 4GB，则其共可存放 $4 \times 1024 \times 1024 \times 1024 / 4 = 1073741824$ 个 int 数据。为 10^9 级别。我假设 e 和 f 都为 10^9 ，所需时间展示如下：



```

Microsoft Visual Studio 调试
e为: 1000000000 f为: 1000000000
第五版代码算法运行时间为: 329519ms
答案为: 30

```

图 5: e 和 f 都为 10^9 的情况下第五版代码所需时间

可以看到即使 e 和 f 都为 10^9 ，我的代码算法都能在有限时间内得出结果。所以对于能在有限时间内处理完的最大规模数据，我最终的第五版代码仍然支持我之前给出的结论：能在有限时间内处理完的最大规模数据取决于内存，只要内存能存放得下数据，我的算法都能在有限时间内处理完。

五、实验经验总结或体会

- 1) 通过这次实验，我掌握了动态规划算法的设计思想，并且学会了如何运用动态规划解决鸡蛋坠落问题。我认识到动态规划算法在解决一类最优化问题时的强大能力，也更加深入地理解了动态规划算法的原理和应用。
- 2) 可以运用蛮力法来验证更优算法的正确性，如在本次实验中我用蛮力法计算得到的结果与第四版代码算法得到的结果对比，发现结果完全一致，通过蛮力法验证了第四版代码算法的正确性。
- 3) 优化算法时可以考虑运用二分算法，看看二分算法是否可以运用到当前问题中来优化算法的时间复杂度。
- 4) 可以考虑运用逆向思维来优化算法。如在本实验中，我就运用了逆向思维反向思考改变了 dp 数组的含义，从而完全改变了动态规划算法的状态转移方程及代码的递推逻辑，成功的降低了算法的时间复杂度。

<p>指导教师批阅意见：</p>	
<p>成绩评定：</p>	
<p>指导教师签字： 年 月 日</p>	
<p>备注：</p>	

<p>指导教师批阅意见：</p>	
<p>成绩评定：</p>	
<p>指导教师签字： 年 月 日</p>	
<p>备注：</p>	

<p>指导教师批阅意见：</p>	
<p>成绩评定：</p>	
<p>指导教师签字： 年 月 日</p>	
<p>备注：</p>	

<p>指导教师批阅意见：</p>	
<p>成绩评定：</p>	
<p>指导教师签字： 年 月 日</p>	
<p>备注：</p>	

<p>指导教师批阅意见：</p>	
<p>成绩评定：</p>	
<p>指导教师签字： 年 月 日</p>	
<p>备注：</p>	

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。