浙江大学

本科生课程报告

学年、学期： 2021 — 2022 学年 秋冬 学期

课程名称： 数据分析与算法设计

任课教师： 李东晓

题　　目： 自选编程作业 4

学生姓名： 黄嘉欣

学　　号： 3190102060

数据分析与算法设计：自选编程作业 4

3190102060 黄嘉欣 信工 1903 班

一、题目：403.青蛙过河(难度：困难)

一只青蛙想要过河。假定河流被等分为若干个单元格，并且在每一个单元格内都有可能放有一块石子(也有可能没有)。青蛙可以跳上石子，但是不可以跳入水中。给你石子的位置列表 stones(用单元格序号升序表示)，请判定青蛙能否成功过河(即能否在最后一步跳至最后一块石子上)。开始时，青蛙默认已站在第 0 块石子上，并可以假定它第一步只能跳跃一个单位(即只能从单元格 1 跳至单元格 2)。如果青蛙上一步跳跃了 k 个单位，那么它接下来的跳跃距离只能选择为 k-1、k 或 k+1 个单位。另请注意，青蛙只能向前(终点的方向)跳跃。各参数满足条件：

- 2 <= stones.length <= 2000
- 0 <= stones[i] <= 2^{31} – 1
- stones[0] == 0

二、算法设计：动态规划

由题意，我们需要判断青蛙能否从 stones[0]开始，在跳跃步数都给定的条件下做到每步都踩上石子，最终到达 stones[n-1]，其中 n=stones.length。

根据题目要求，若青蛙上一步跳跃了 k 个单位，它接下来的跳跃距离只有 k-1、k 或 k+1 三个选择。假设青蛙已经到达了第 i 块石子，则由上述条件我们可以反推出青蛙上一次起跳时脚踩石子的位置，此即为状态的递推关系。注意，此时不能正向递推，因为我们并不知道在距第 i 块石子 k-1、k 或 k+1 个单位处是否有石子存在。因此，令 F(i,k)表示青蛙能否从某块石子跳 k 个单位到达第 i 块石子，则 k=stones[i]-stones[j]，其中 stones[j]为上一次起跳位置，j<i。根据跳跃距离的要求，由于青蛙从第 j 块石子跳到第 i 块石子时跳跃了 k 个单位，则当青蛙从某个位置跳到第 j 块石子时，跳跃距离只能从 k-1、k 或 k+1 个单位中选择(反向推导)。因此，若要使青蛙能从第 j 块石子跳 k 个单位到达第 i 块石子，即 F(i,k)=1，必须要 F(j,k-1)||F(j,k)||F(j,k+1)=1，以保证其能先从某块石子跳到第 j 块石子；否则，若 F(j,k-1)||F(j,k)||F(j,k+1)=0，青蛙将无法到达 stones[j]，方案不合法，有 F(i,k)=0。故状态转移方程为：

F(i,k)=F(j,k-1)||F(j,k)||F(j,k+1), 0<=j<i<n, k=stones[i]-stones[j]

考虑初始情况，若青蛙身处在第 0 块石子，只跳跃 0 个单位，则必然能够到达第 0 块石子，故有 F(0,0)=1。综上，完整的状态转移方程为：

$$\begin{cases} F(i,k) = F(j,k-1)||F(j,k)||F(j,k+1), 0 \le j < i < n, k = stones[i] - stones[j] \\ F(0,0) = 1 \end{cases}$$

考虑到或运算的特殊性，只要 F(j,k-1)、F(j,k)或 F(j,k+1)中有一个值为 1，即可确定 F(i,k)=1。因此，我们需要将矩阵元素全部初始化为 0，再将初始情况填入。由于 j<i，可以采用按行上往下的填充方法填表。因为 k 不一定连续，故表不一定会被填充完整。在程序的最后，我们需要判断矩阵的最后一列 F(n-1,k)(k 为任意值)中是否有值为 1，以说明青蛙能够成功到达第 n-1 块石子。

当然，根据题目条件，我们可以推导出一些限制条件，以提高算法效率。当青蛙位于第 0 块石子上时，其上一次跳跃距离只能为 0，之后每次跳跃，青蛙所在的石子编号至少加 1，而跳跃距离至多加 1，因此，当跳跃 m 次后，青蛙所在石子编号 i>=m，而上一次跳跃距离 k<=m，故 k<=i。结合递推关系的推导过程，若青蛙已位于第 j 块石子，其上次跳跃最多只能跳跃 j 个单位，则从第 j 块石子跳到第 i 块石子最多只能跳跃 j+1 个单位，故有限制条件 k<=j+1。通过从后向前穷举上一次所在石子编号 j，我们可以判断各变量是否满足此不等式，若不满足，便可直接开始下次循环，从而减少循环次数，提高算法的时间效率。此算法的伪代码为：

```
算法：canCross(int* stones, int stonesSize)
      // 判断青蛙能否到达最后一块石子
      // 输入：每块石子的位置，石子的总数量
      // 输出：bool 值，若能到达则为 true，否则为 false
      // 注意：F(,)为 stonesSize*stonesSize 的全 0 数组，不再额外定义
      F(0,0) <- 1 // 初始情况
      for i <- 1 to stonesSize-1 do
          for j <- i-1 to 0 do
              k <- stones[i]-stones[j]
              if k > j+1 do // 限制条件
                  break
              F(i,k) <- (F(j,k-1)||F(j,k)||F(j,k+1)) // 递推，填充矩阵
      for k <-0 to stonesSize-1 do
          if F(stonesSize-1,k) == 1 do
              return true // 最后一列有 1，表明能到达 stones[stonesSize-1]
      return false // 无法到达
```

三、代码实现

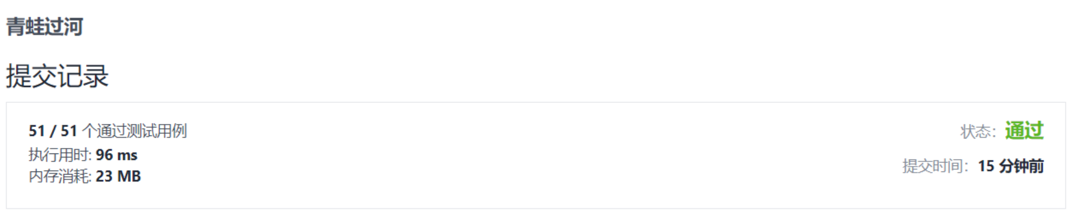
根据(二)中伪代码，具体化各步骤，可得 C 语言代码实现为：

代码：

```c
bool canCross(int* stones, int stonesSize)
{
    int F[stonesSize][stonesSize];
    int i, j, k;
    for (i=0;i<stonesSize;i++){
        for (j=0;j<stonesSize;j++){
            F[i][j] = 0; // 二维数组初始化
        }
    }
    F[0][0] = 1; // 初始条件
    for (i=1;i<stonesSize;i++) { // 注意起始点
        for (j=i-1;j>=0;j--) {
            k = stones[i]-stones[j]; // 跳跃距离
            if (k>j+1) { // 限制条件
                break;
            }
            F[i][k] = F[j][k-1]||F[j][k]||F[j][k+1]; // 填充
        }
    }
    for (k=0;k<stonesSize;k++){
        if (F[stonesSize-1][k] == 1)
            return true; // 能到达最后一块石子
    }
    return false; // 无法到达
}
```

四、运行结果

如图 4.1，将动态规划算法代码提交至 LeetCode，得其执行用时为 96ms，内存消耗 23MB，通过全部 51 个测试用例。当测试输入为[0,1,2,3,4,8,9,11]时，程序输出为 false，与正确结果一致，如图 4.2 所示。采用自测试实例，若输入的数据为[0,1,2,4,5,7,11,16]，则青蛙可通过石子 stones[0]->stones[1]->stones[2]->stones[3]->stones[5]->stones[6]->stones[7]到达河对岸，如图 4.3 所示，程序输出正确。对于更多的输入可能，无法一一列举，但由 LeetCode 测试结果可知，算法设计正确。



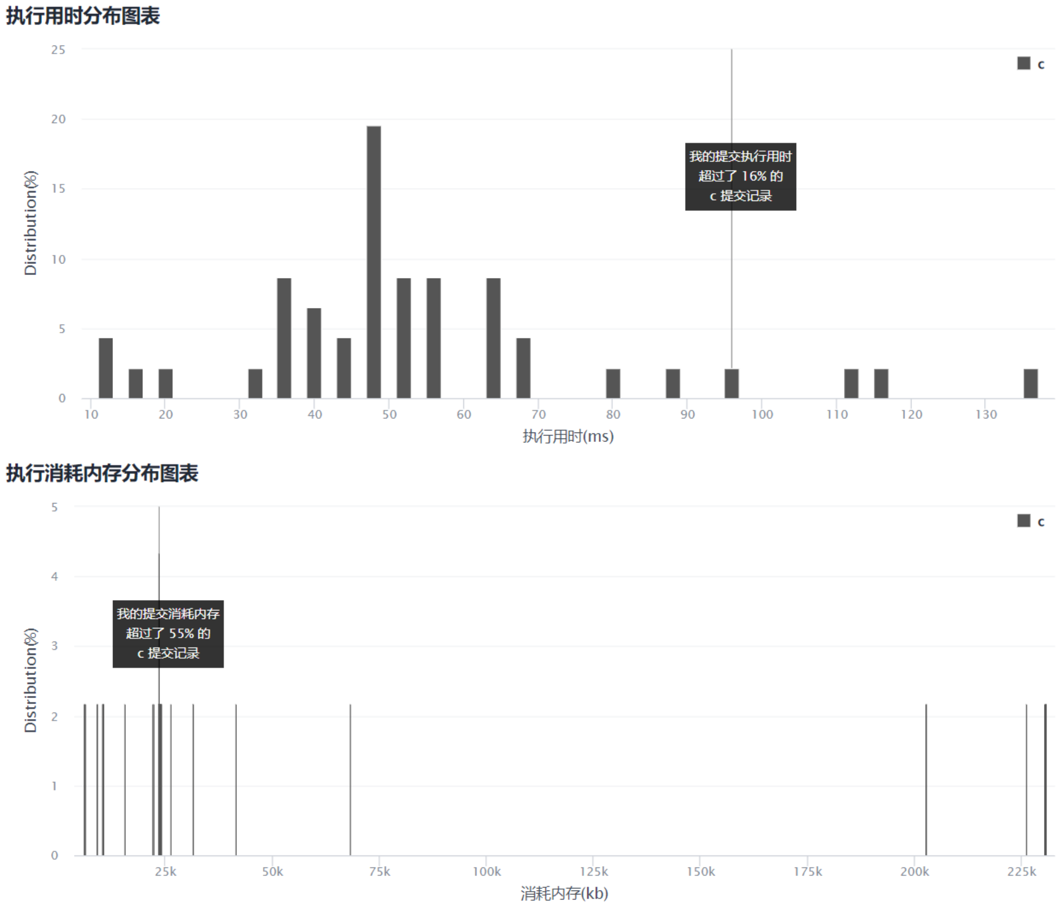


图 4.1 LeetCode 算法运行结果

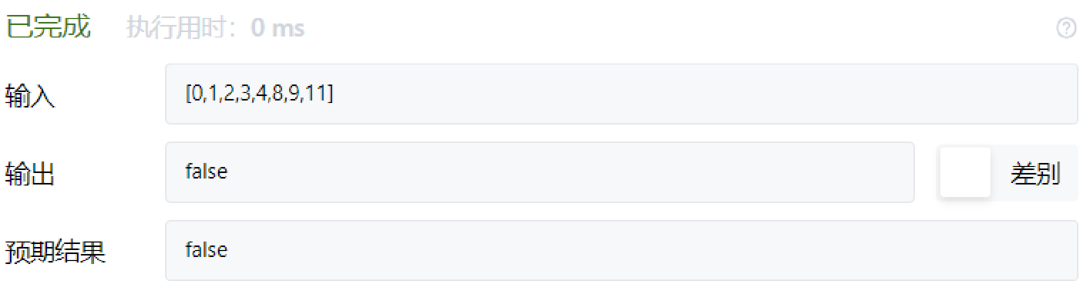


图 4.2 LeetCode 测试实例

图 4.3 自测试实例

五、效率分析

由代码，设对长度为 n 的数组 stones[n]，算法的基本操作（赋值）执行次数为 C(n)，则在最差情况下，即限制条件始终满足时，有：

$$C(n)=\sum_{i=1}^{n}\sum_{j=0}^{i-1}(1+1)=2\sum_{i=1}^{n}i=2\frac{n(n+1)}{2}\in O(n^2)$$

故动态规划算法的最差时间效率为 $O(n^2)$，此问题属于 P 类问题。

显然，由于在动态规划算法中，我们只需要对二维数组 F[n][n]进行填充，故所需的额外空间为：$V(n)=n^2\in O(n^2)$，即算法的空间效率为 $O(n^2)$。

六、总结

总的来说，此题的设计思路较为复杂，需要理清青蛙跳跃的前后关系，并从题目已知中推导出限制条件，从而得到更好的算法。本质上来说，青蛙过河问题利用了路径可逆的性质，将问题进行了等效对偶：表面上我们在正向递推，但实际上是在验证是否存在某条反向路径从第 n-1 块石子到达第 0 块石子，思路十分巧妙。

根据我的经验，一般来说，动态规划问题的解题步骤可以分为以下五步：①确定子问题；②确定状态；③推导状态转移方程；④确定边界条件；⑤算法优化。对动态规划问题来说，如何理清变量关系、找到状态转移方程是十分重要的一件事情。除此之外，对边界条件的确定、算法的优化，都需要具体情况具体分析，从题目的限制条件中推出。尽管如此，动态规划算法能够有效帮助我们解决许多问题，学会从新的角度出发，往往能够让我们走出思维定式，在解题时“豁然开朗”。