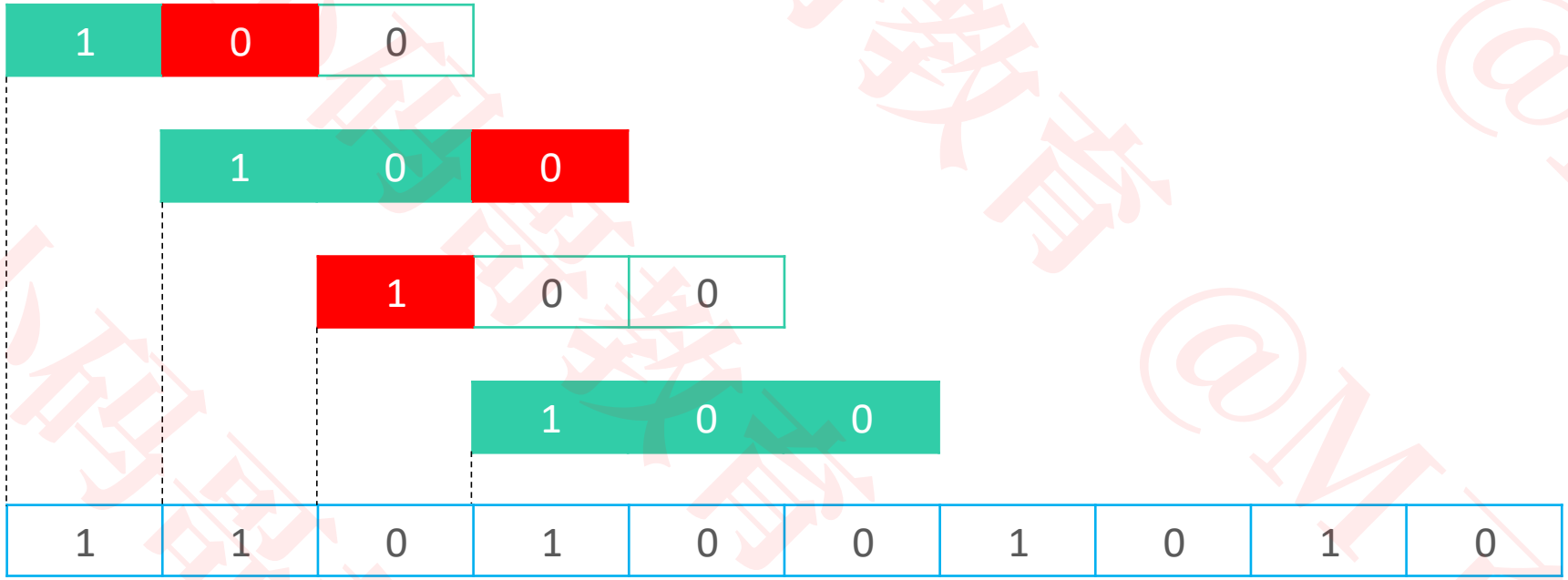
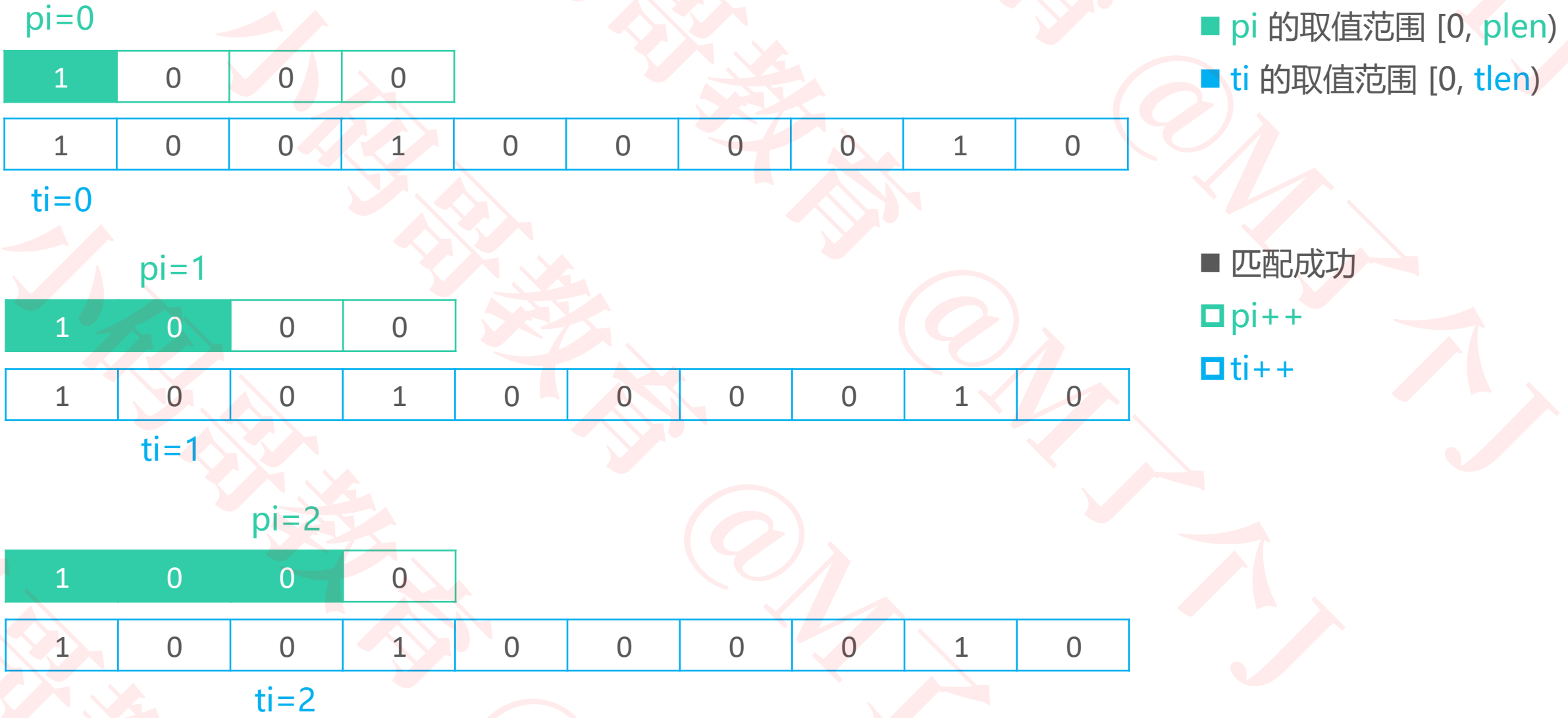
蛮力（Brute Force）

- 以字符为单位，从左到右移动模式串，直到匹配成功



- 蛮力算法有 2 种常见实现思路

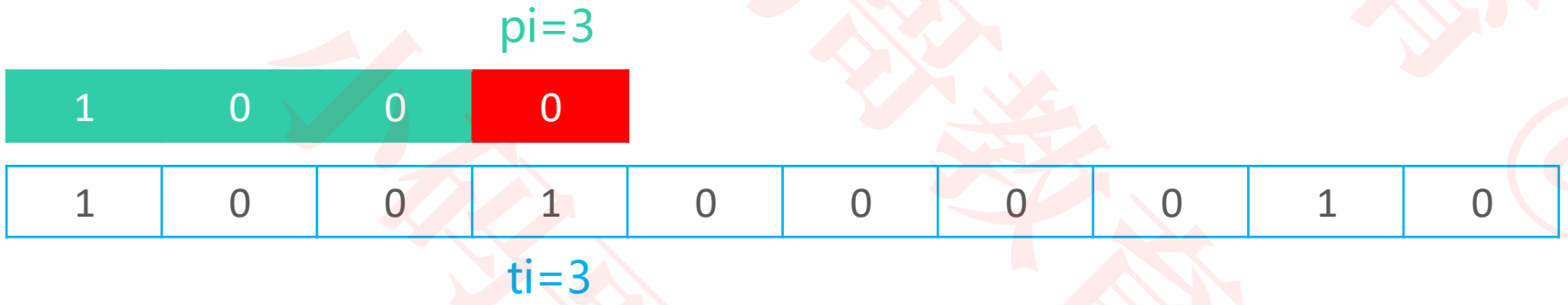
蛮力1 – 执行过程



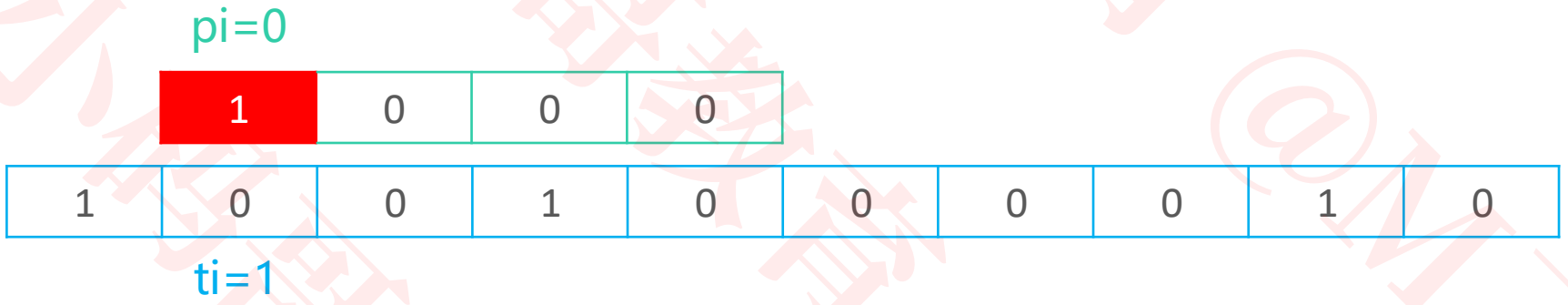
- pi 的取值范围 [0, plen)
- ti 的取值范围 [0, tlen)

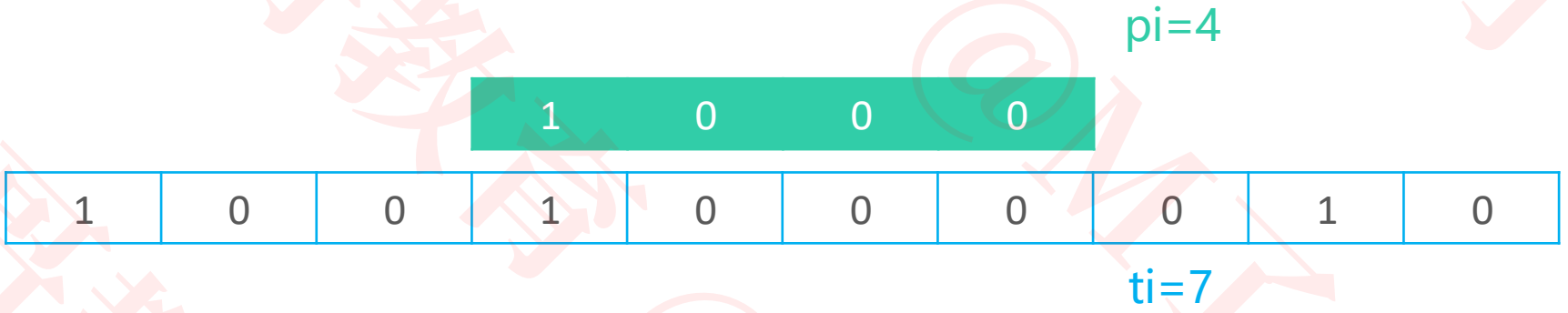
- 匹配成功
- pi++
- ti++

蛮力1 – 执行过程



- 匹配失败
  - pi = 0
  - ti –= pi – 1





pi == plen

代表匹配成功

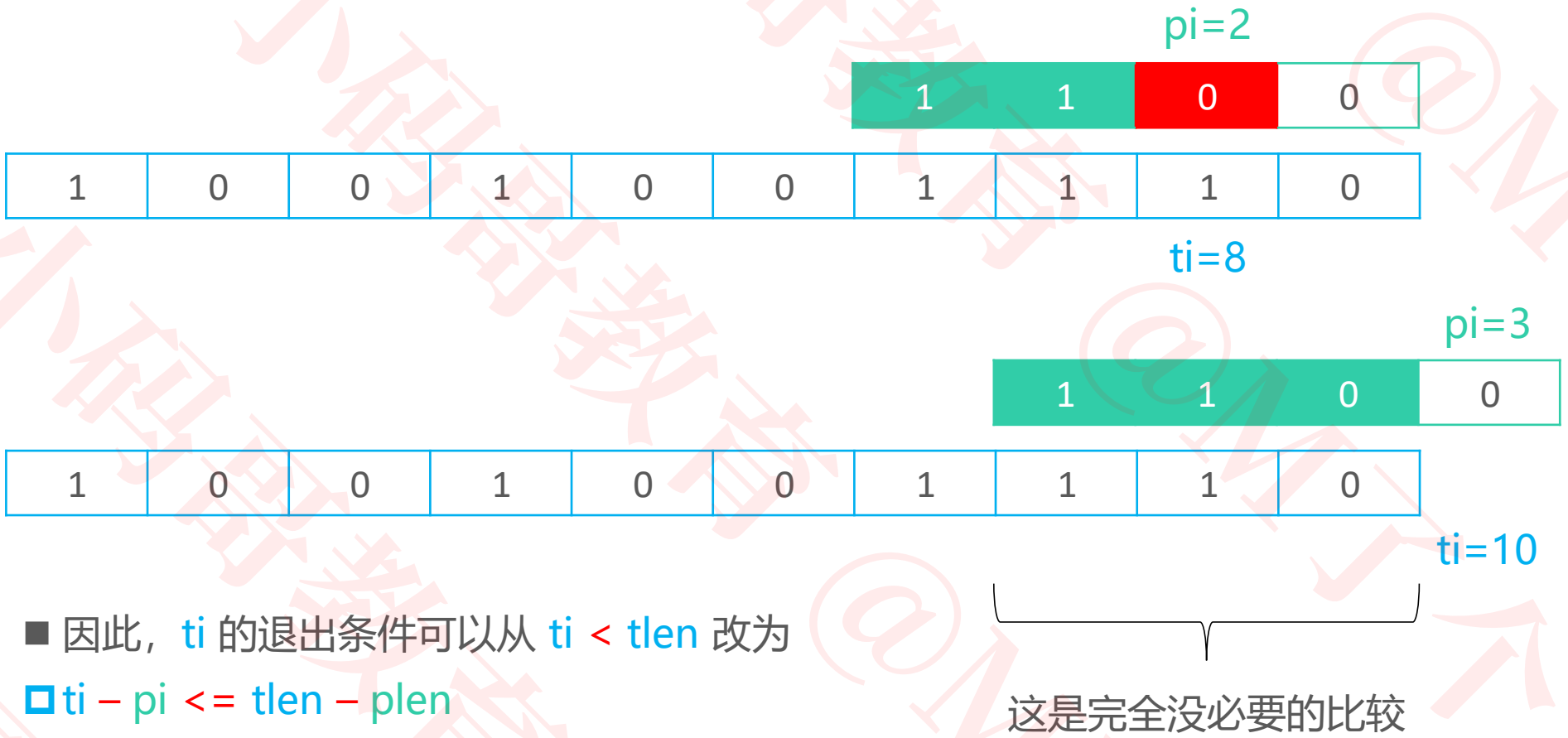
小码哥教育 SEEMYGO

蛮力1 – 实现

```
public static int indexOf(String text, String pattern) {
    if (text == null || pattern == null) return -1;
    int tlen = text.length();
    int plen = pattern.length();
    if (tlen == 0 || plen == 0 || tlen < plen) return -1;
    int pi = 0, ti = 0;
    while (pi < plen && ti < tlen) {
        if (text.charAt(ti) == pattern.charAt(pi)) {
            ti++;
            pi++;
        } else {
            ti -= pi - 1;
            pi = 0;
        }
    }
    return pi == plen ? ti - pi : -1;
}
```

蛮力1 – 优化

- 此前实现的蛮力算法，在恰当的时候可以提前退出，减少比较次数



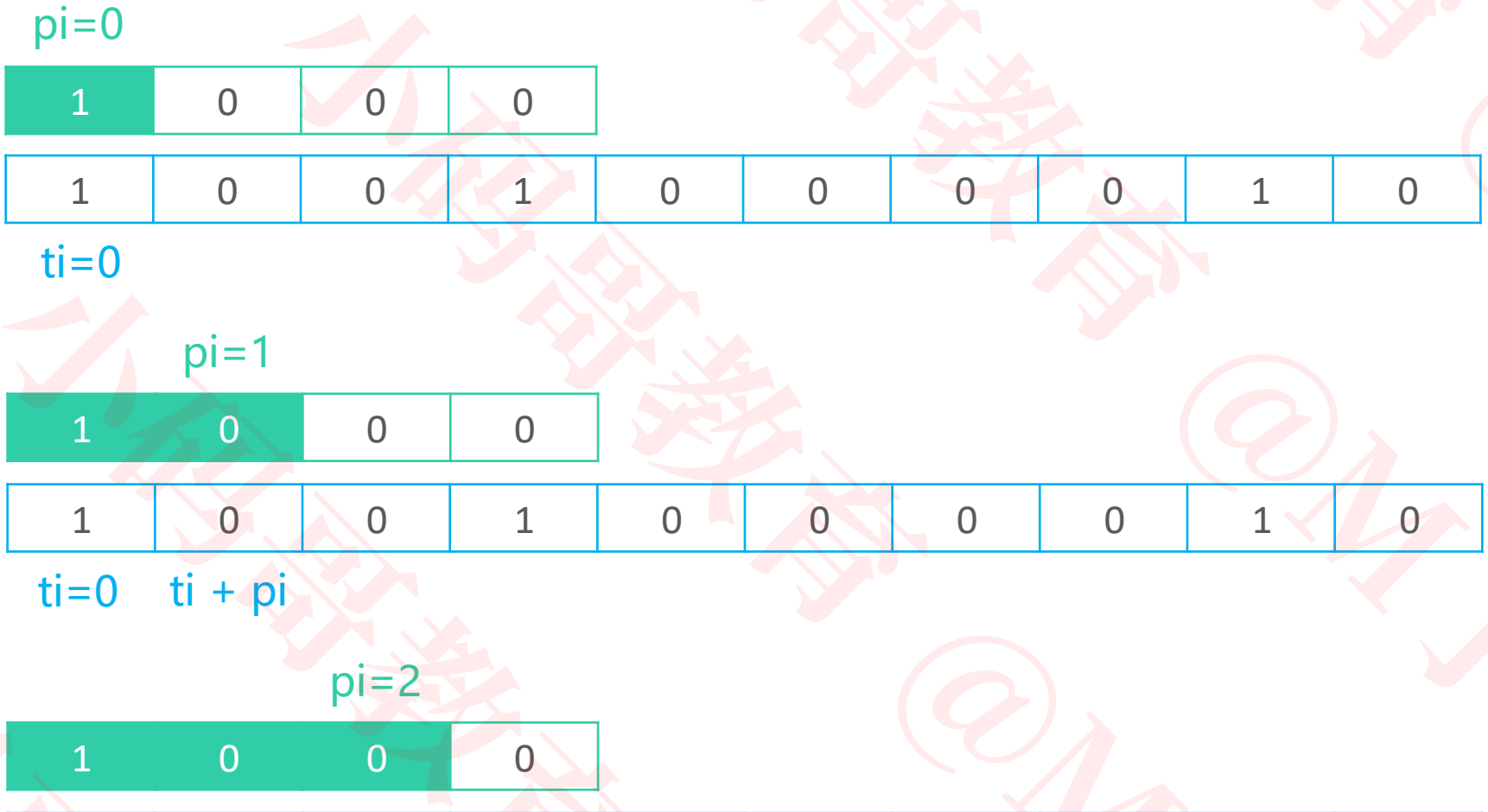
- 因此，ti 的退出条件可以从 ti < tlen 改为
  - ti – pi <= tlen – plen
  - ti – pi 是指每一轮比较中 Text 首个比较字符的位置

小码哥教育 SEEMYGO

蛮力1 – 优化实现

```
public static int indexOf(String text, String pattern) {
    if (text == null || pattern == null) return -1;
    int tlen = text.length();
    int plen = pattern.length();
    if (tlen == 0 || plen == 0 || tlen < plen) return -1;
    int pi = 0, ti = 0;
    int tmax = tlen - plen;
    while (pi < plen && ti - pi <= tmax) {
        if (text.charAt(ti) == pattern.charAt(pi)) {
            ti++;
            pi++;
        } else {
            ti -= pi - 1;
            pi = 0;
        }
    }
    return pi == plen ? ti - pi : -1;
}
```

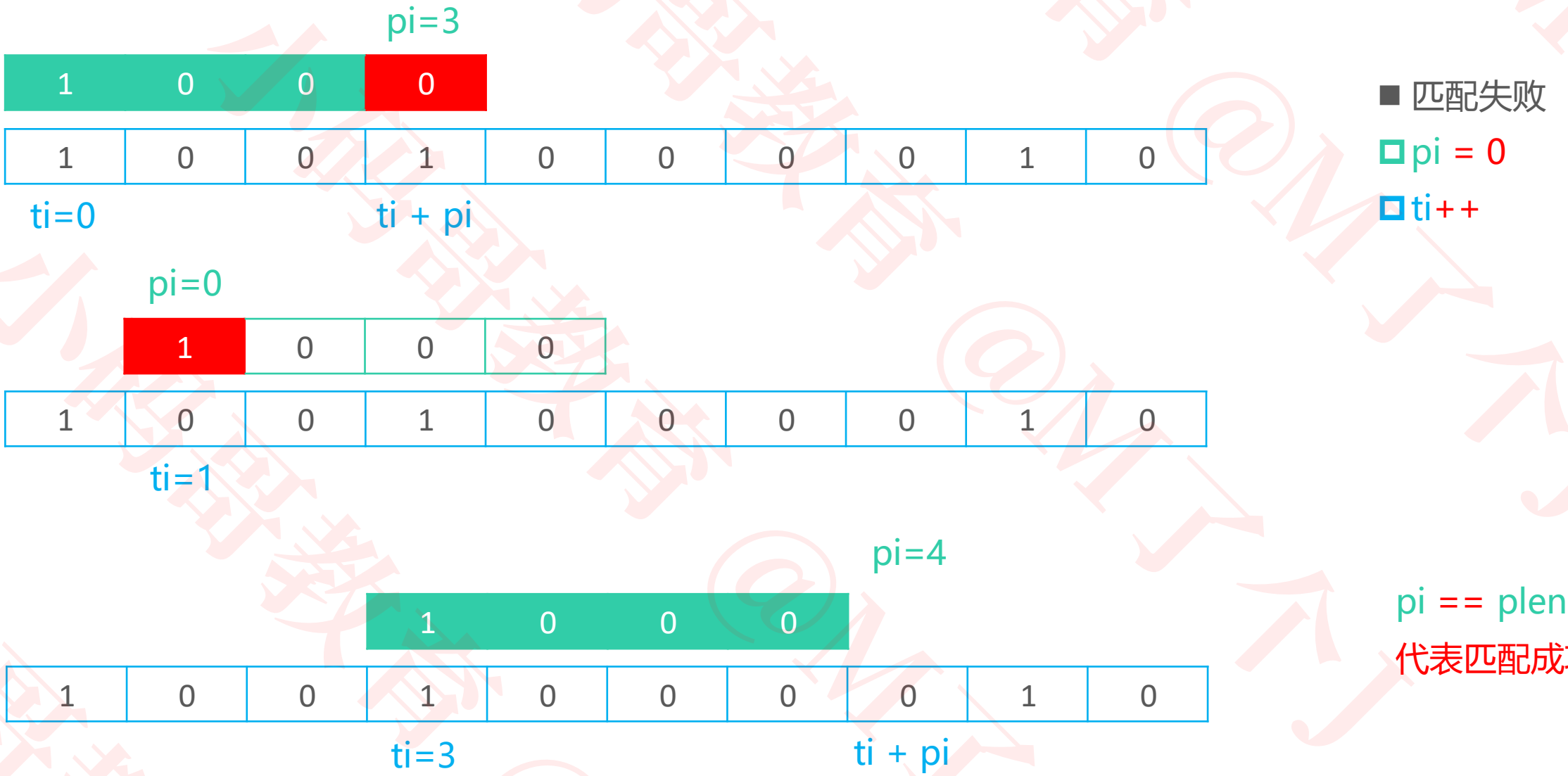
蛮力2 – 执行过程



- pi 的取值范围 [0, plen)
- ti 的取值范围 [0, tlen – plen]

- ti 是指每一轮比较中 Text 首个比较字符的位置

蛮力2 – 执行过程



■ 匹配失败

pi = 0

ti++

pi == plen

代表匹配成功

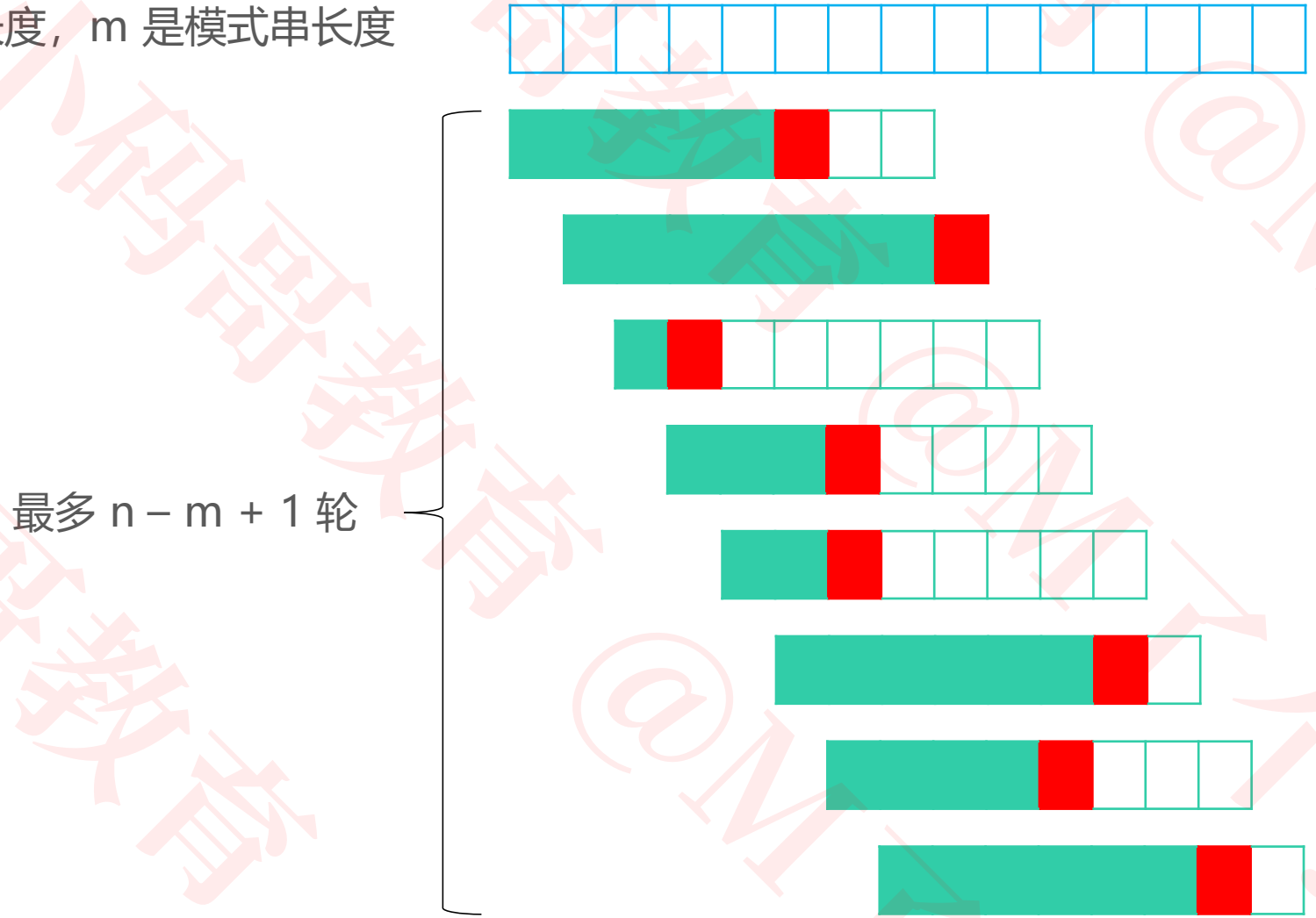
小码哥教育 SEEMYGO

蛮力2 – 实现

```
public static int indexOf(String text, String pattern) {
    if (text == null || pattern == null) return -1;
    int tlen = text.length();
    int plen = pattern.length();
    if (tlen == 0 || plen == 0 || tlen < plen) return -1;
    int tmax = tlen - plen;
    for (int ti = 0; ti <= tmax; ti++) {
        int pi = 0;
        for (; pi < plen; pi++) {
            if (text.charAt(ti + pi) != pattern.charAt(pi)) break;
        }
        if (pi == plen) return ti;
    }
    return -1;
}
```

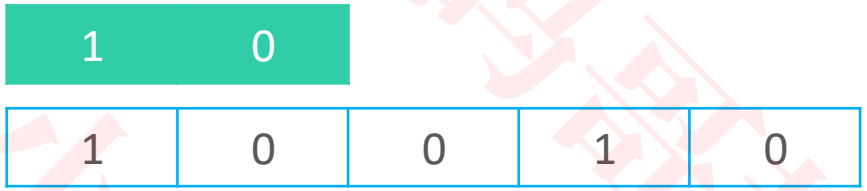
蛮力 – 性能分析

- n 是文本串长度，m 是模式串长度



蛮力 – 性能分析

- 最好情况
  - 只需一轮比较就完全匹配成功，比较 m 次（ m 是模式串的长度）
  - 时间复杂度为 $O(m)$



- 最坏情况（字符集越大，出现概率越低）
  - 执行了 n – m + 1 轮比较（ n 是文本串的长度）
  - 每轮都比较至模式串的末字符后失败（ m – 1 次成功，1 次失败）
  - 时间复杂度为 $O(m * (n - m + 1))$，由于一般 m 远小于 n，所以为 $O(nm)$