算法与数据结构体系课程

liuyubobobo

字符串匹配

最朴素的算法

字符串匹配最朴素的算法

字符串匹配

Hello, this is liuyubobobo

bo

源字符串 s

目标字符串 t

字符串匹配最朴素的算法

字符串匹配

Hello, this is liuyubobobo

bo

20

源字符串 s

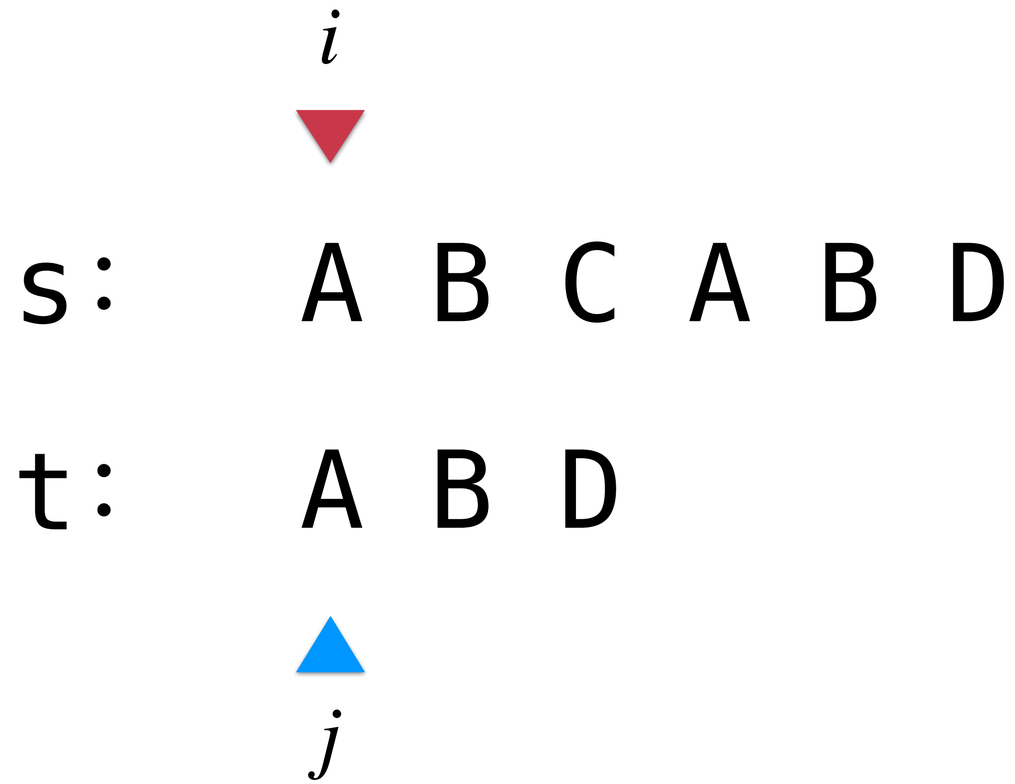
目标字符串 t

返回源字符串在目标字符串首次出现的索引

如果没有找到，返回 -1

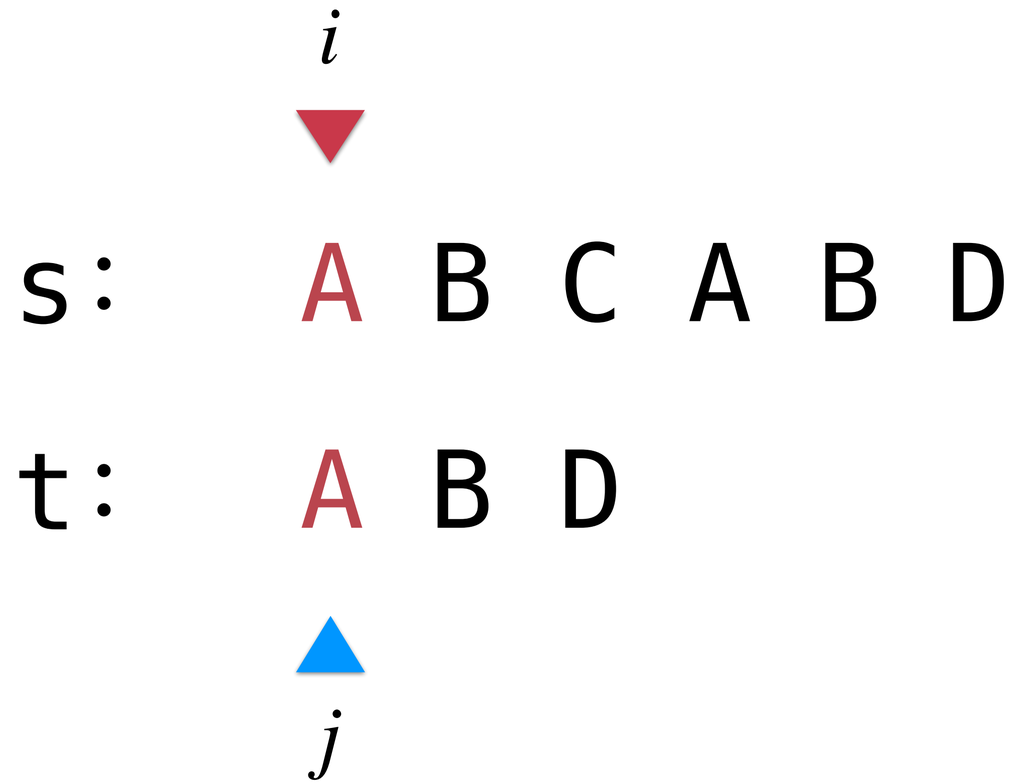
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



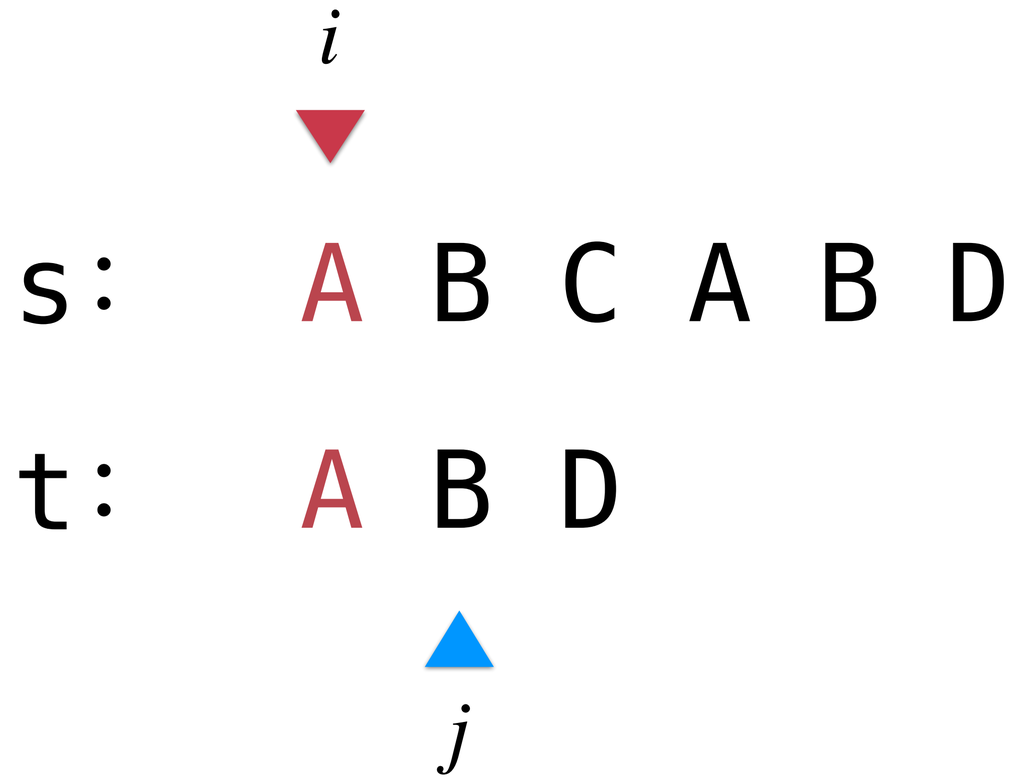
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



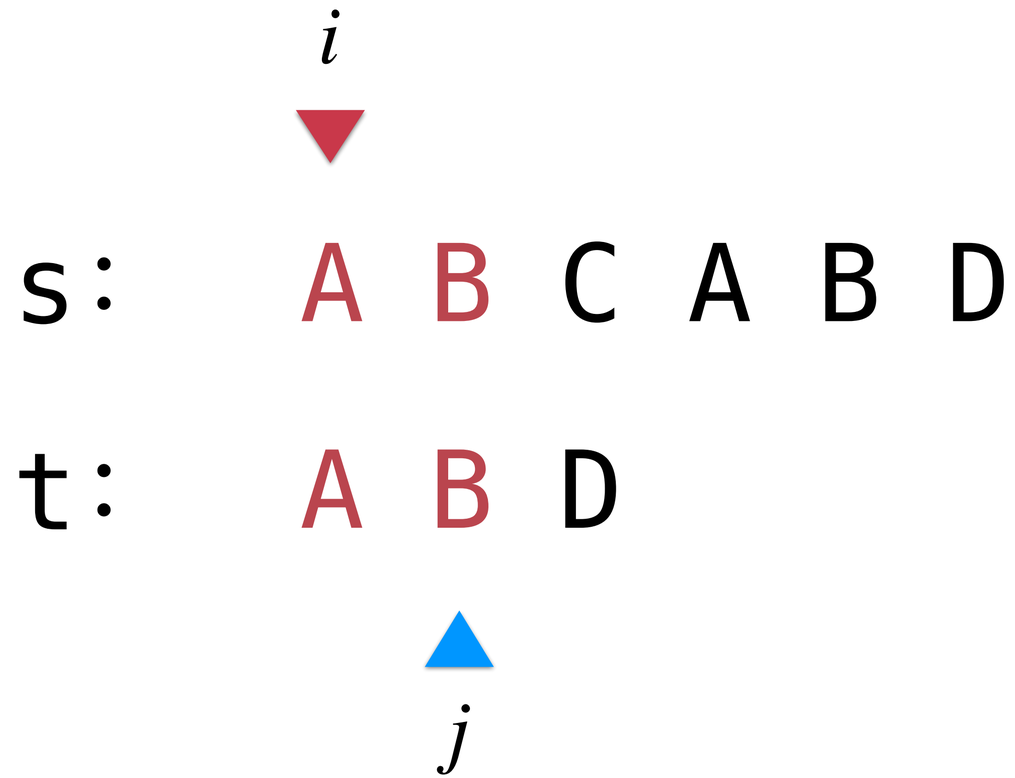
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



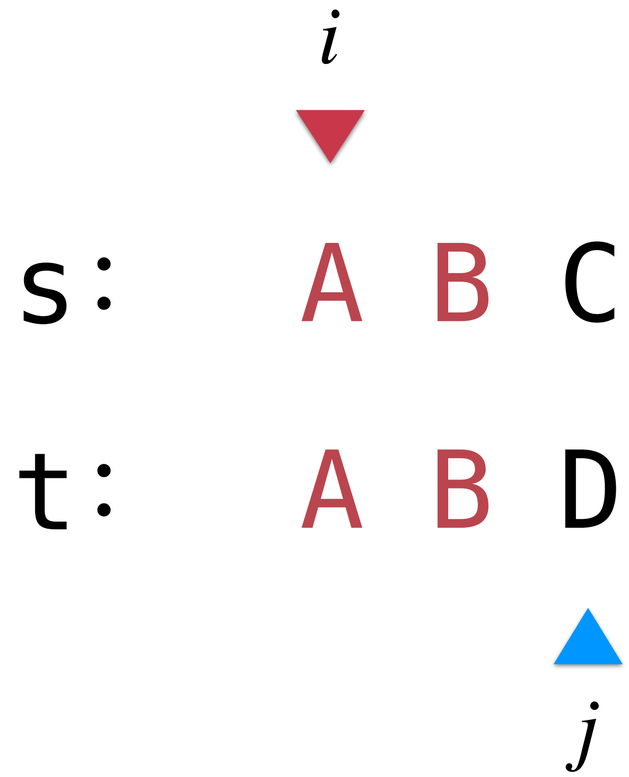
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



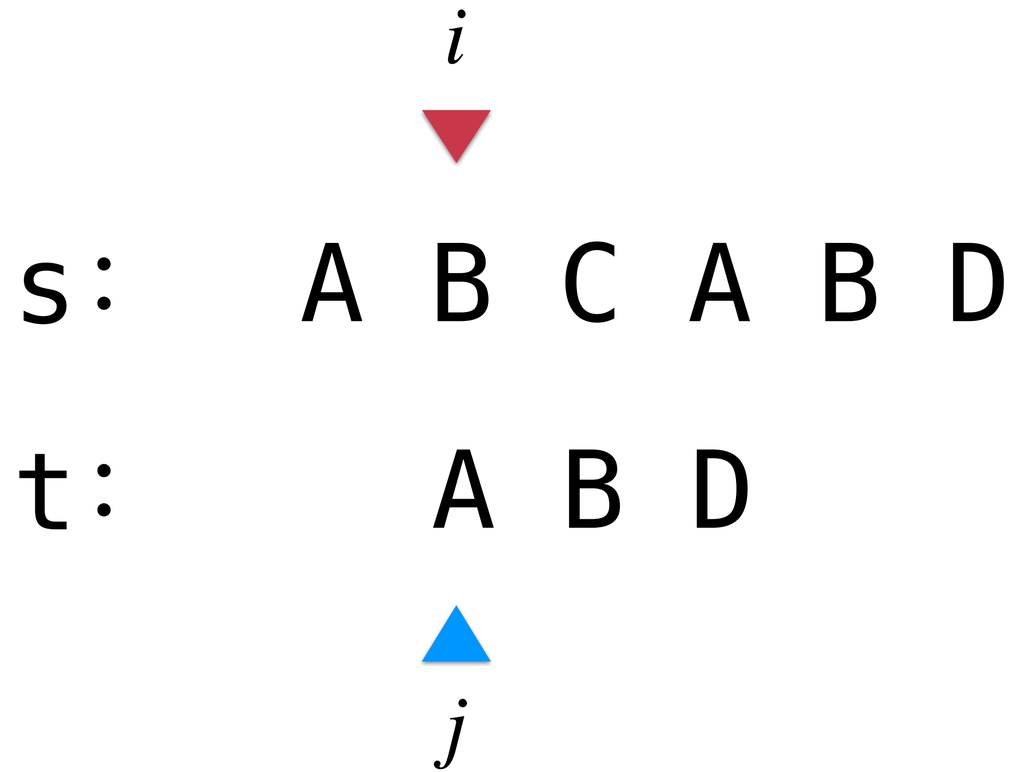
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



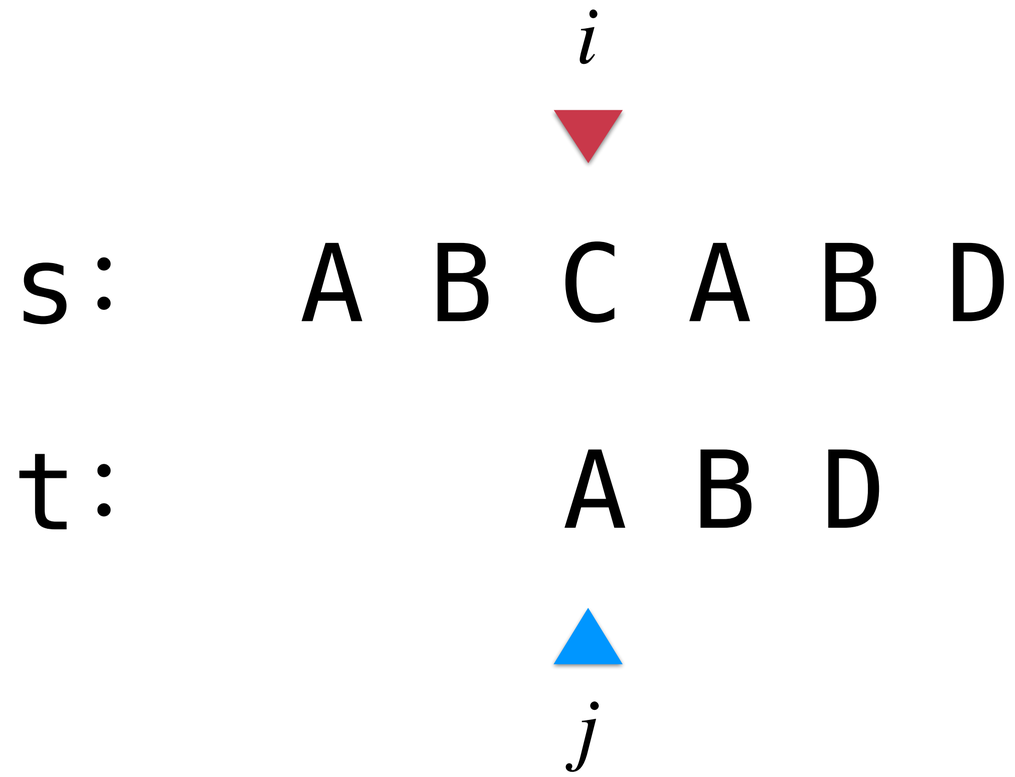
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



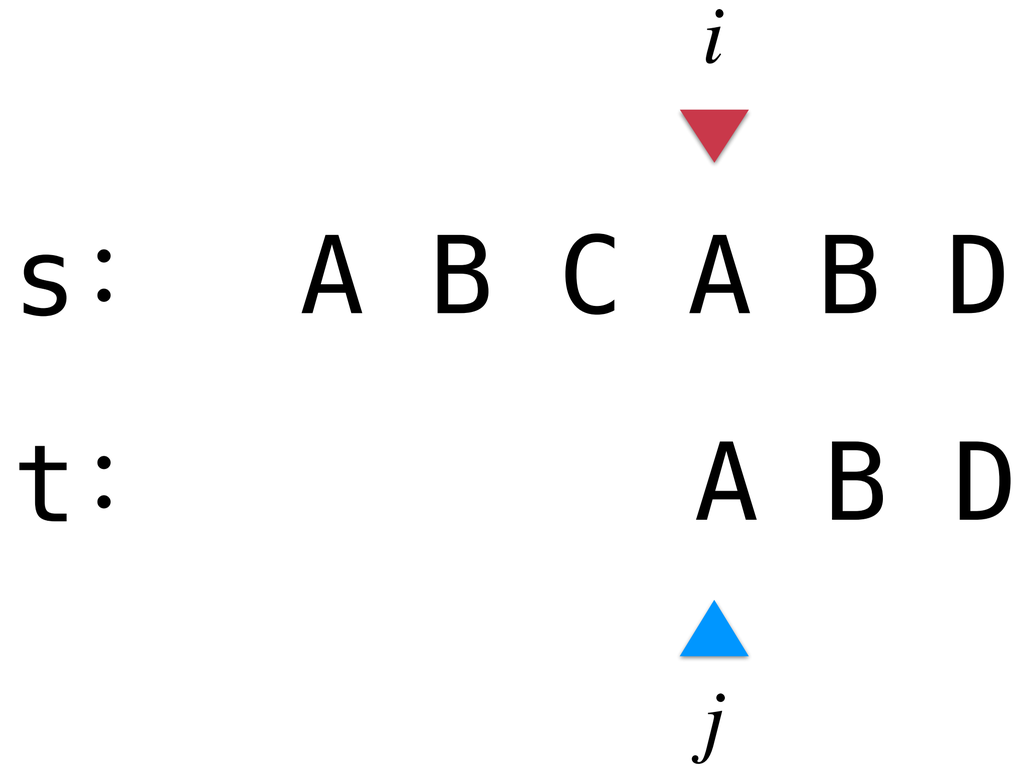
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



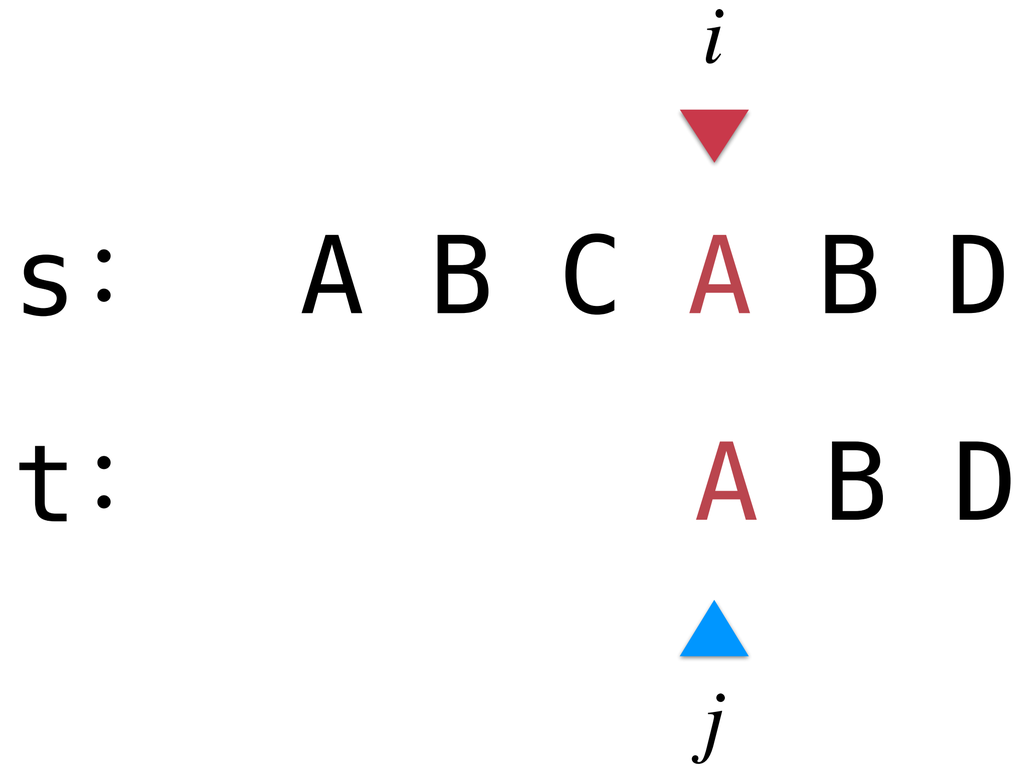
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



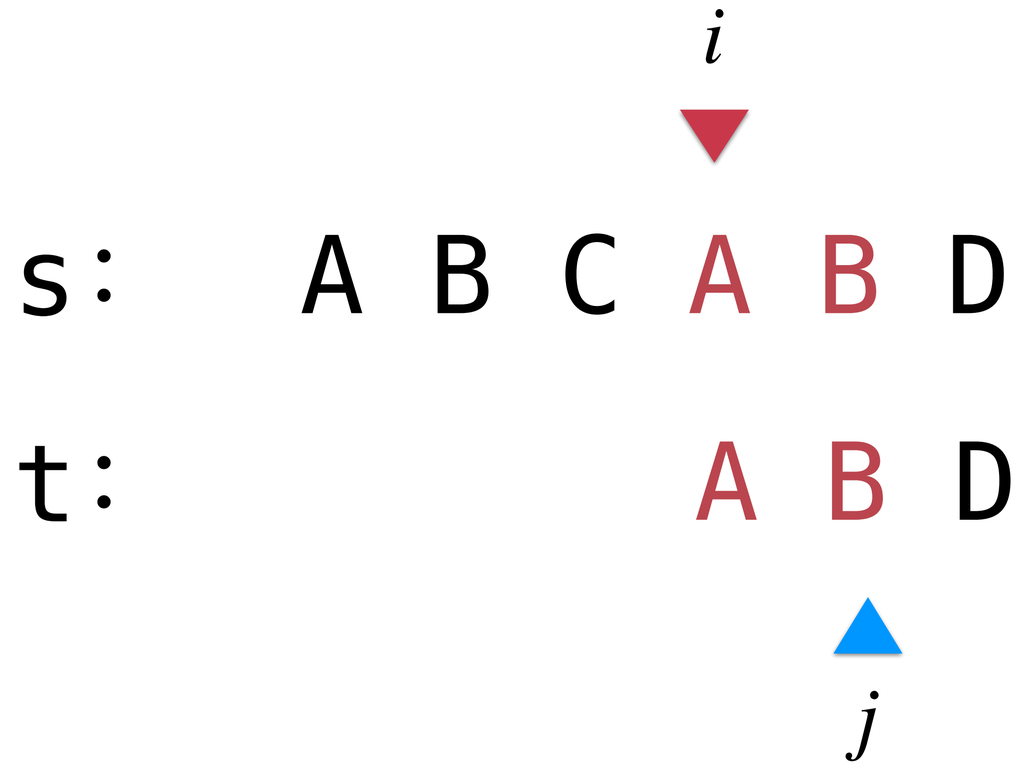
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



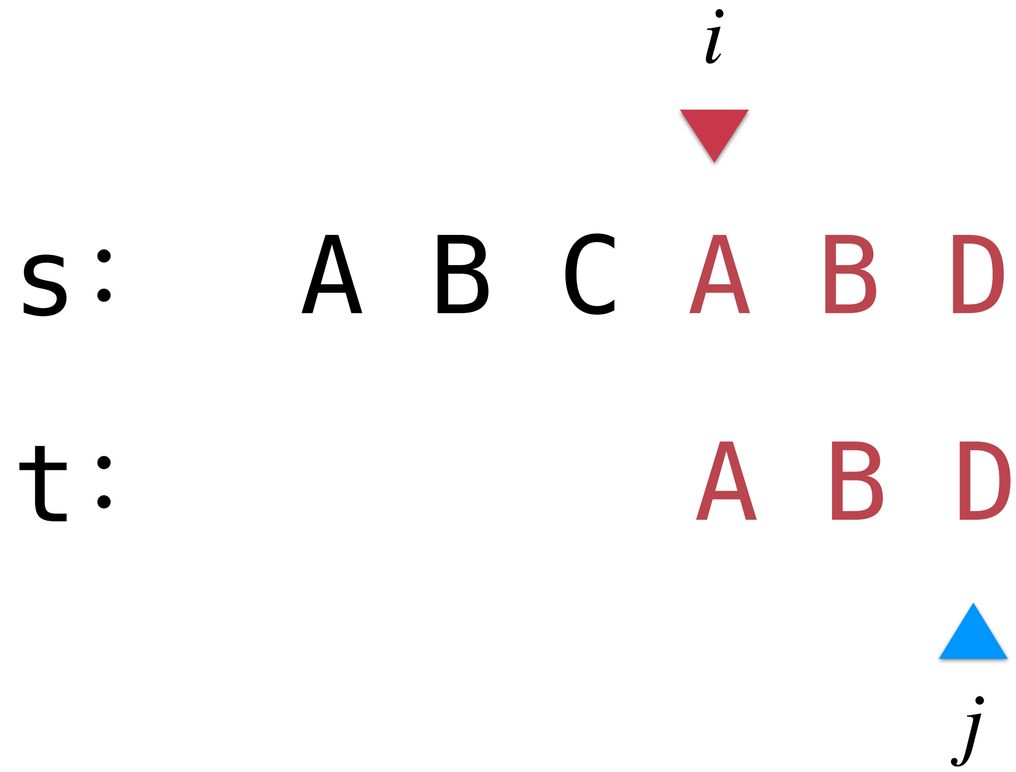
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



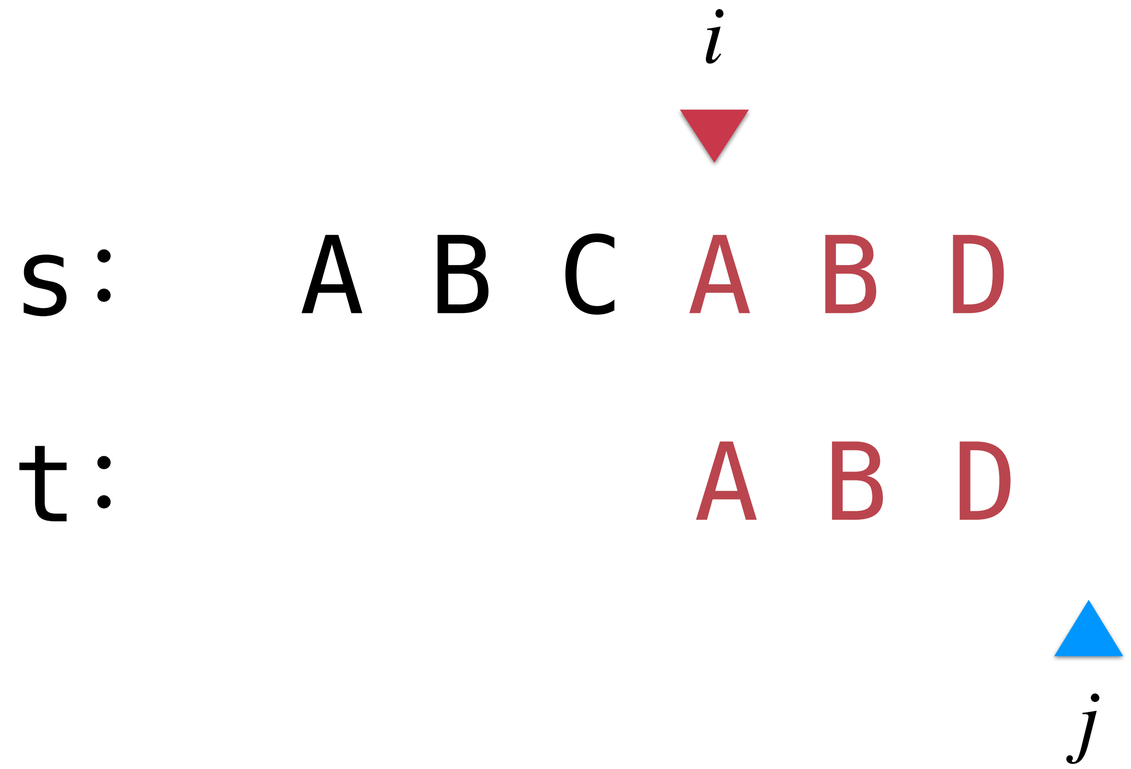
字符串匹配最朴素的算法

字符串匹配　　暴力搜索



字符串匹配最朴素的算法

字符串匹配　　暴力搜索



对每一个 s 的起始点 i

尝试匹配 t

最差：|s| * |t|

实际上匹配 t 的过程可以提前终止

返回 i

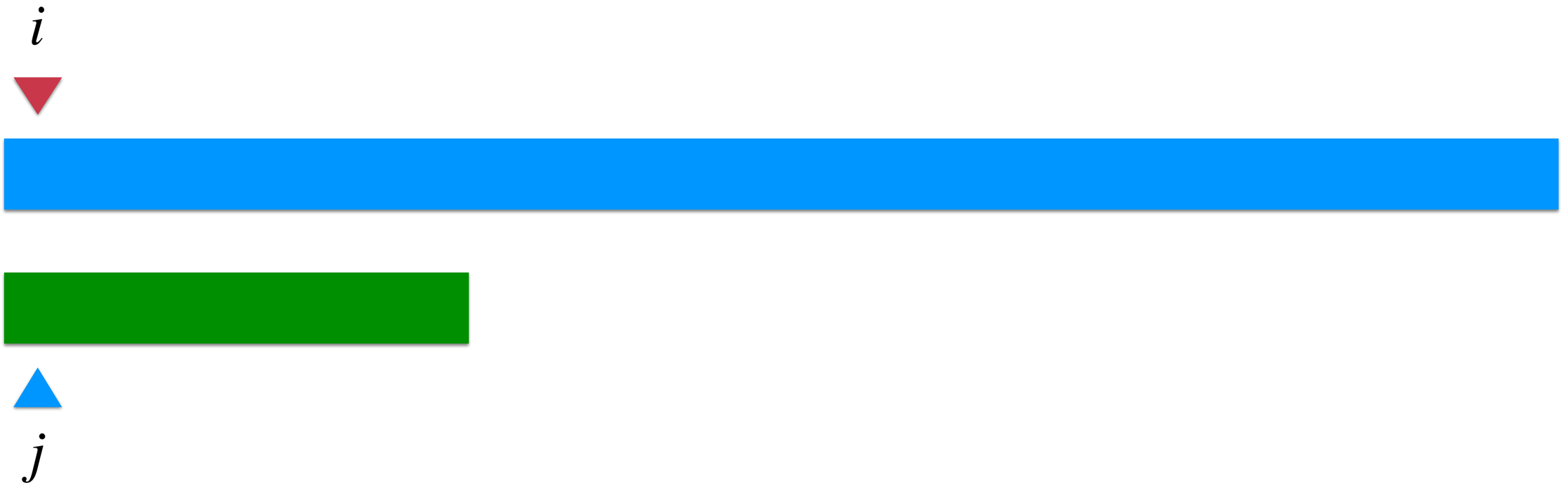
实现字符串的暴力匹配

实践：实现字符串的暴力匹配

实践：字符串暴力匹配的性能分析

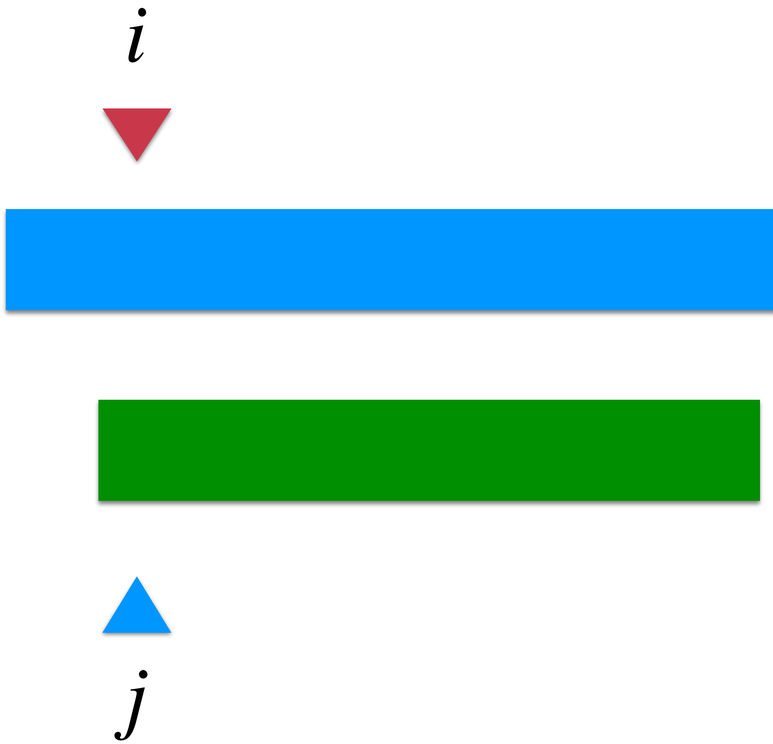
哈希法

字符串暴力匹配的改进思路



字符串暴力匹配的改进思路

字符串暴力匹配的改进思路



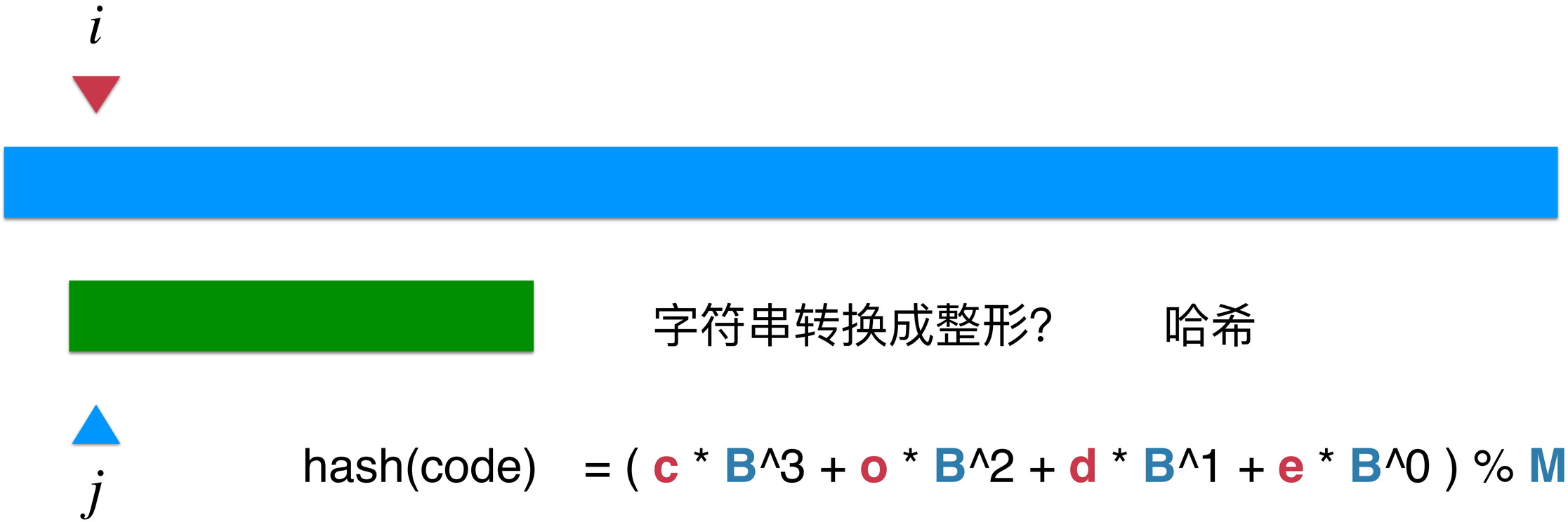
每一次匹配，最差情况，重新扫描一遍匹配串 t

比较两个字符串是否相等是 O(n) 的

比较两个整型是否相等是 O(1) 的

字符串转换成整形?　　哈希

字符串暴力匹配的改进思路



hash(code) = (**c** * **B**^3 + **o** * **B**^2 + **d** * **B**^1 + **e** * **B**^0) % **M**

字符串暴力匹配的改进思路

字符串转换成整形?　　哈希

hash(code)　= (**c** * **B**^3 + **o** * **B**^2 + **d** * **B**^1 + **e** * **B**^0) % **M**

hash(code)　　= (((**c** * **B** + **o**) % **M** * **B** + **d**) % **M** * **B** + **e**) % **M**

```
int hash = 0
for(int i = 0 ; i < s.length() ; i ++)
    hash = (hash * B + s.charAt(i)) % M
```

字符串暴力匹配的改进思路

扔掉匹配问题，看字符串转哈希思想的一个应用

Leetcode 1147

看问题：Leetcode 1147

https://leetcode-cn.com/problems/longest-chunked-palindrome-decomposition/

Leetcode 1147

Leetcode 1147

Leetcode 1147

Leetcode 1147

每次添加一个字符，重新匹配?

把字符串转换成哈希值，每次比较哈希值

每添加一个字符，新的哈希值的计算

Leetcode 1147

把字符串转换成哈希值，每次比较哈希值

每添加一个字符，新的哈希值的计算

前面： hashcode * B + newchar　　123 -> 1234　　123 * 10 + 4

后面： newchar * B^ (len - 1) + hashcode　　123 -> 4123　　4 * (10^3) + 123

实现 Leetcode 1147

实践：Leetcode 1147

Leetcode 1392

实践：暴力求解 Leetcode 1392

使用哈希法求解 Leetcode 1392

使用哈希法求解 Leetcode 1392

前面： hashcode * B + newchar　　123 -> 1234　　123 * 10 + 4

后面： newchar * B^ (len - 1) + hashcode　　123 -> 4123　　4 * (10^3) + 123

使用哈希法求解 Leetcode 1392

前面： hashcode * B + newchar　　123 -> 1234　　123 * 10 + 4

后面： newchar * B^ (len - 1) + hashcode　　123 -> 4123　　4 * (10^3) + 123

使用哈希法求解 Leetcode 1392

前面： hashcode * B + newchar　　123 -> 1234　　123 * 10 + 4

后面： newchar * B^ (len - 1) + hashcode　　123 -> 4123　　4 * (10^3) + 123

(x % M) / B != (x / B) % M

1230 % 11 / 10 = 9 / 10

1230 / 10 % 11 = 123 % 11 = 2

1234 -> 123　　1234 -> (1234 - 4) / 10

使用哈希法求解 Leetcode 1392

(a + b) % M == (a % M + b % M) % M

(a * b) % M == (a % M * b % M) % M

(a / b) % M != ((a % M) / (b % M)) % M

(x % M) / B != (x / B) % M

1230 % 11 / 10 = 9 / 10

1230 / 10 % 11 = 123 % 11 = 2

1234 -> 123　　1234 -> (1234 - 4) / 10

使用哈希法求解 Leetcode 1392

从短字符串的哈希计算到长字符串的哈希　　把中间的计算结果储存起来

(a / b) % M != ((a % M) / (b % M)) % M

(x % M) / B != (x / B) % M

1230 % 11 / 10 = 9 / 10

1230 / 10 % 11 = 123 % 11 = 2　　1234 -> 123　　1234 -> (1234 - 4) / 10

实践：使用哈希法求解 Leetcode 1392

Leetcode 187

实践：直接使用哈希表求解 Leetcode 187

滚动哈希求解 Leetcode 187

滚动哈希求解 Leetcode 187

滚动哈希求解 Leetcode 187

滚动哈希求解 Leetcode 187

滚动哈希求解 Leetcode 187

滚动哈希求解 Leetcode 187

滚动哈希求解 Leetcode 187

因为每个字符只有 ACGT 四种取值

A->1, C->2, G->3, T->4

长度为 10 的字符串的哈希值，可以用一个 10 位的数字表示

long

滚动哈希求解 Leetcode 187

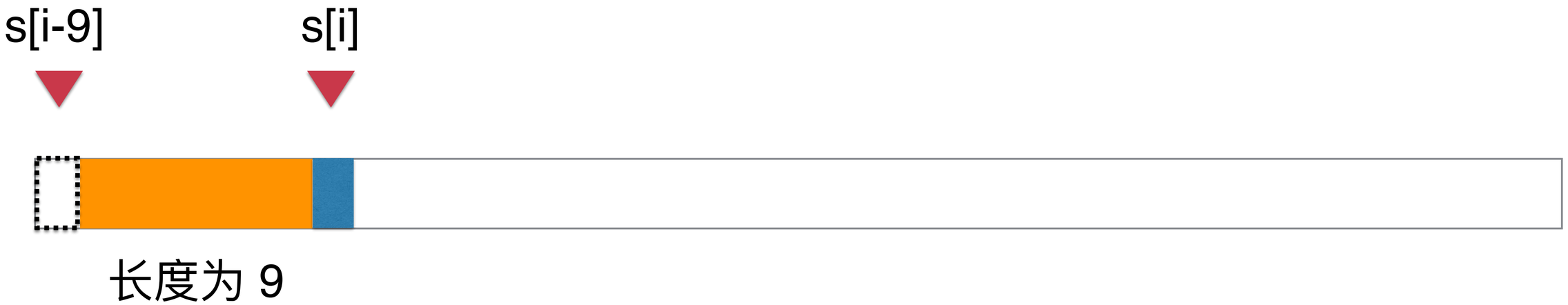
长度为 9

hash = hash * 10 + map[s[i]]

此时 hash 是一个长度为10的字符串的哈希值

123412341 * 10 + 2 = 1234123412

滚动哈希求解 Leetcode 187



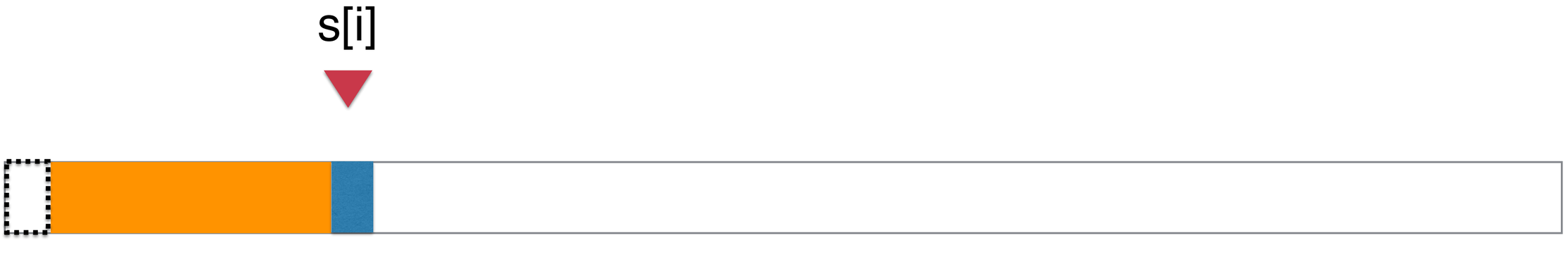
hash = hash * 10 + map[s[i]]

此时 hash 是一个长度为10的字符串的哈希值

hash = hash - map[s[i-9]] * (10^9)

1234123412 - 1000000000 = 234123412

滚动哈希求解 Leetcode 187



hash = hash * 10 + map[s[i]]

此时 hash 是一个长度为10的字符串的哈希值

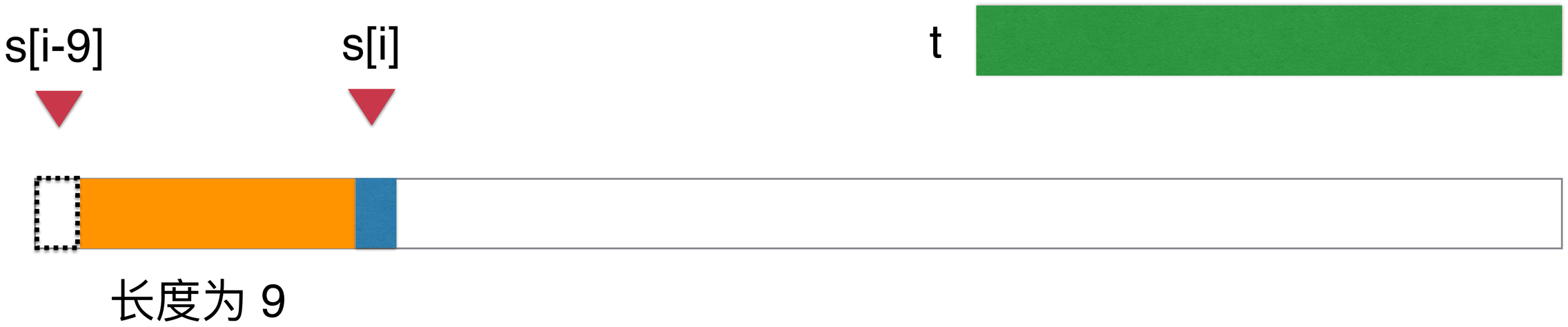
hash = hash - map[s[i-9]] * (10^9)

1234123412 - 1000000000 = 234123412

实践：滚动哈希求解 Leetcode 187

使用滚动哈希求解字符串匹配问题

使用滚动哈希求解字符串匹配问题



hash = hash * 10 + map[s[i]]

此时 hash 是一个长度为10的字符串的哈希值

hash = hash - map[s[i-9]] * (10^9)

使用滚动哈希求解字符串匹配问题

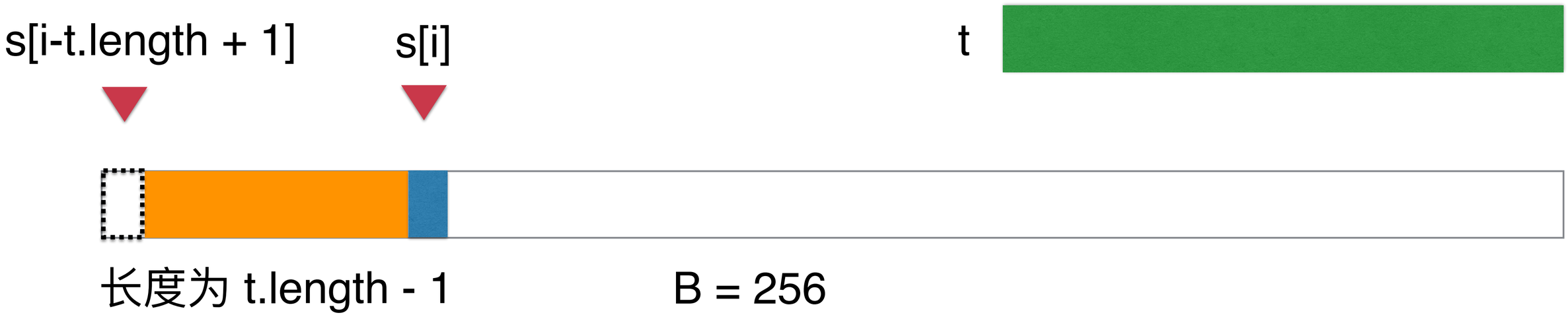
长度为 t.length - 1

hash = hash * 10 + map[s[i]]

此时 hash 是一个长度为10的字符串的哈希值

hash = hash - map[s[i-9]] * (10^9)

使用滚动哈希求解字符串匹配问题

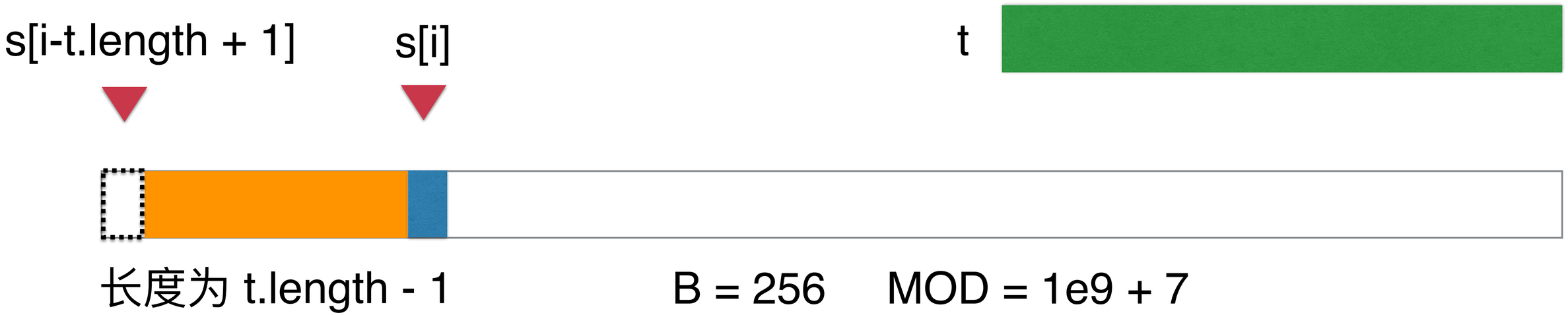


hash = hash * 10 + map[s[i]]

此时 hash 是一个长度为10的字符串的哈希值

hash = hash - map[s[i-9]] * (10^9)

使用滚动哈希求解字符串匹配问题



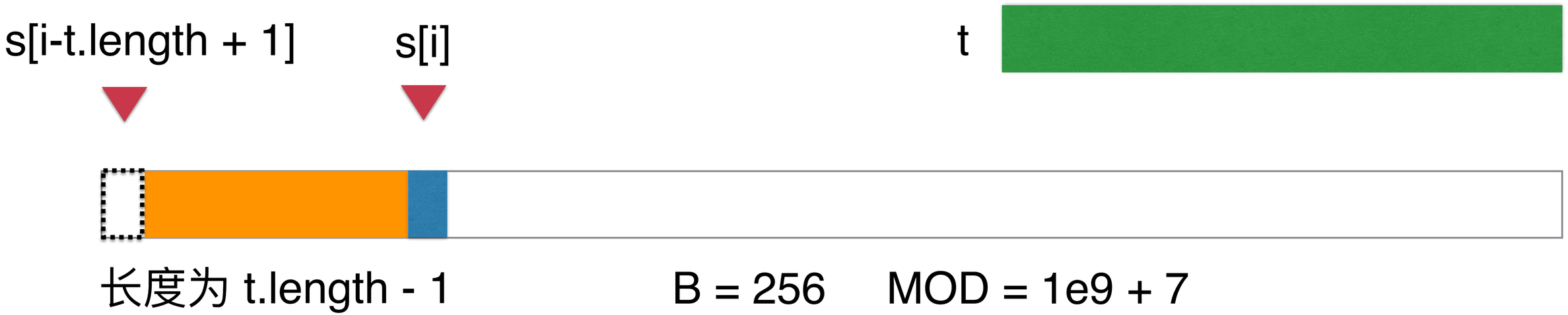
长度为 t.length - 1　　B = 256　　MOD = 1e9 + 7

hash = (hash * B + s[i]) % MOD

此时 hash 是一个长度为10的字符串的哈希值

hash = hash - map[s[i-9]] * (10^9)

使用滚动哈希求解字符串匹配问题



长度为 t.length - 1　　B = 256　　MOD = 1e9 + 7

hash = (hash * B + s[i]) % MOD

此时 hash 是一个长度为 t.length 的字符串的哈希值

hash = hash - map[s[i-9]] * (10^9)

使用滚动哈希求解字符串匹配问题

长度为 t.length - 1　　B = 256　　MOD = 1e9 + 7

hash = (hash * B + s[i]) % MOD

此时 hash 是一个长度为 t.length 的字符串的哈希值

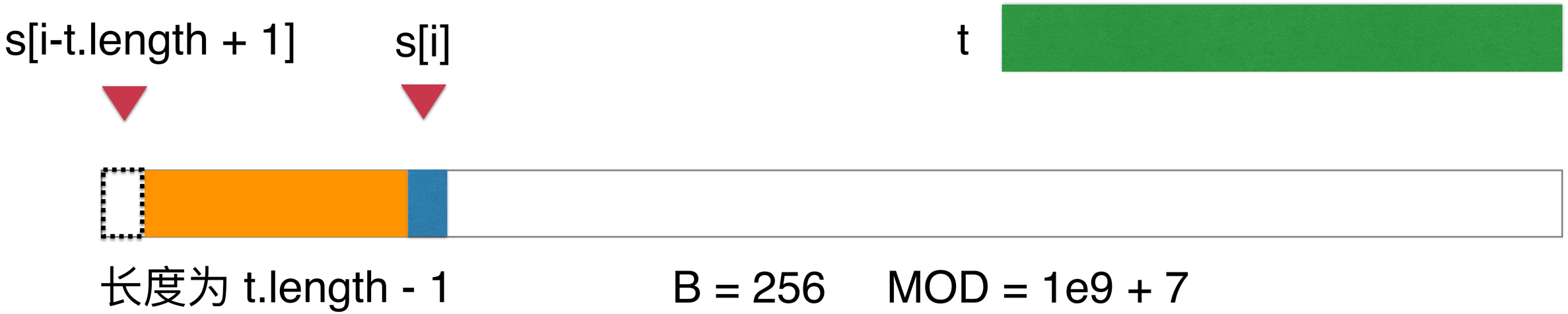
hash = hash - s[i-t.length + 1] * (B^(t.length - 1)) %MOD +MOD

3 点向前 8 个小时

3 - 8 = -5

-5 + 12 = 7 点

使用滚动哈希求解字符串匹配问题



长度为 t.length - 1　　B = 256　　MOD = 1e9 + 7

hash = (hash * B + s[i]) % MOD

Rabin-Karp 算法

此时 hash 是一个长度为 t.length 的字符串的哈希值

hash = (hash - s[i-t.length + 1] * (B^(t.length - 1)) % MOD + MOD) % MOD

实践：实现 Rabin-Karp 算法

Rabin-Karp 算法的性能分析

Rabin-Karp 算法的性能分析

O(1) 获得每一个 t.length 的子串的哈希值

先比较哈希值，只有哈希值相等，才处理哈希冲突

最差情况，每一步都哈希冲突　　O(mn)

如果要寻找每一个子串的话：

aaaaaaaaaaaaaaaaaaaaa
aaaaaa

Rabin-Karp 算法的性能分析

O(1) 获得每一个 t.length 的子串的哈希值

先比较哈希值，只有哈希值相等，才处理哈希冲突

最差情况，每一步都哈希冲突　　O(mn)

如果要寻找每一个子串的话：

aaaaaaaaaaaaaaaaaaaa

　aaaaaa

其他

欢迎大家关注我的个人公众号：是不是很酷

算法与数据结构体系课程

liuyubobobo