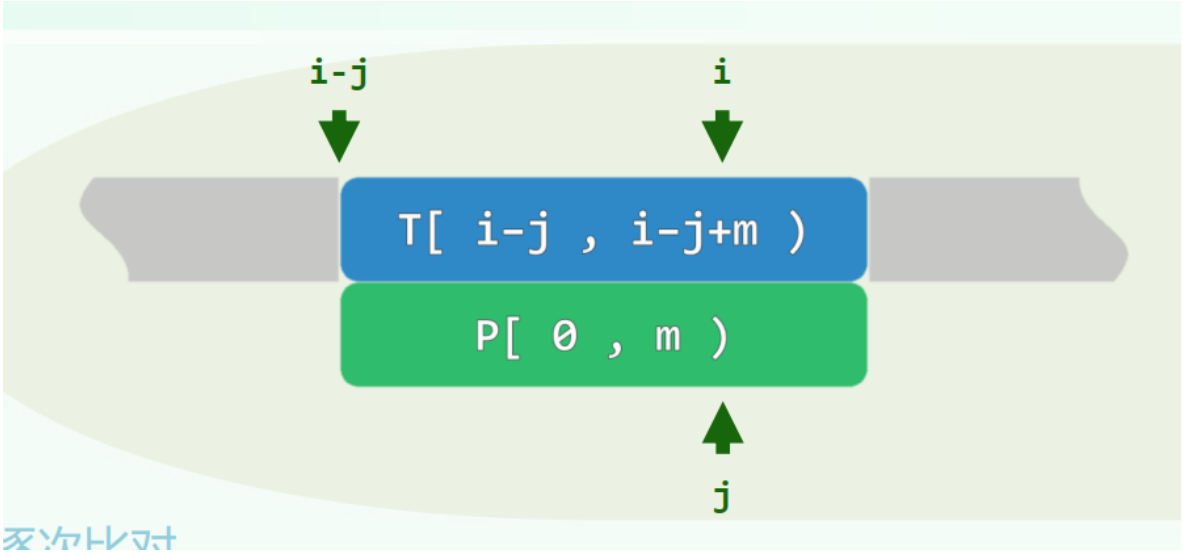
串——模式匹配问题

蛮力算法

版本 1

自左到右逐次比对

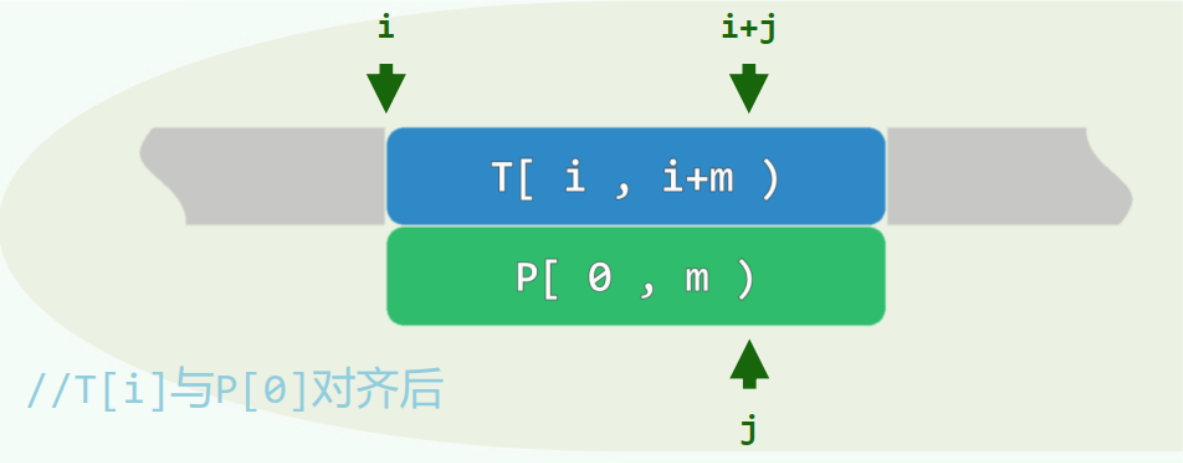
- 若匹配转到下一个字符；
- 否则，T 回退，P复位。



版本 2

`T[i]` 和 `P[0]` 对齐之后

- 逐次比对
  - 失配，转到下一对齐位置



返回值：最终对齐位置

KMP算法：记忆法

D.E.Knuth, J.H.Morris, V.R.Pratt

原理

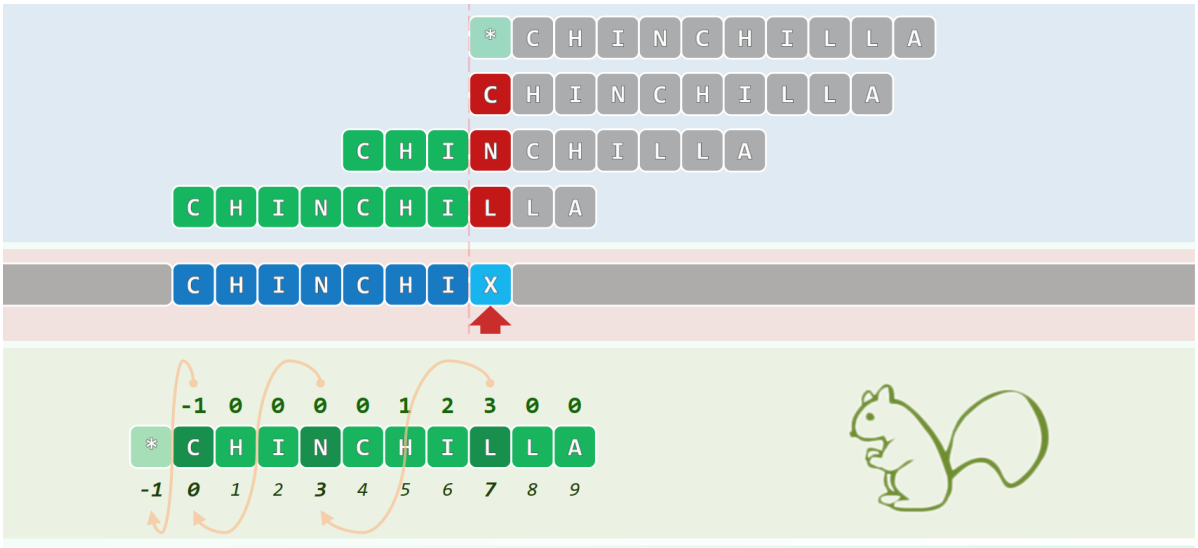
对于上面的版本1，能否少回退一些?

因为只想要完全匹配，所以事实上我们只要考虑**还可能**达成完全匹配的最靠左侧的位置即可。

跳转表：next

这个的意思，就是模式串 `P` 每个位置对应的前缀的 最长公共前后缀。

$next[0] = -1$，指向通配符



时间复杂度分析

$k = 2i - j$ 单调不减，最后达到 $2(n-1) - (-1) = 2n - 1$

为什么单调不减：因为每次匹配上 `i++` 且 `j++` 的时候， `k` 恰好增加 1； 失配的时候 `j = next[j]` 会使得 `k` 至少增加 1。

根据习题 11-4 ， 这里的 `i` 就是成功的比对次数，而 `i-j` 不小于失败的匹配次数。

改进：减少回跳次数

next 考虑下一个位置不匹配的才能进入 next 位置。

这个尤其在字符集非常密集的时候会很有用。

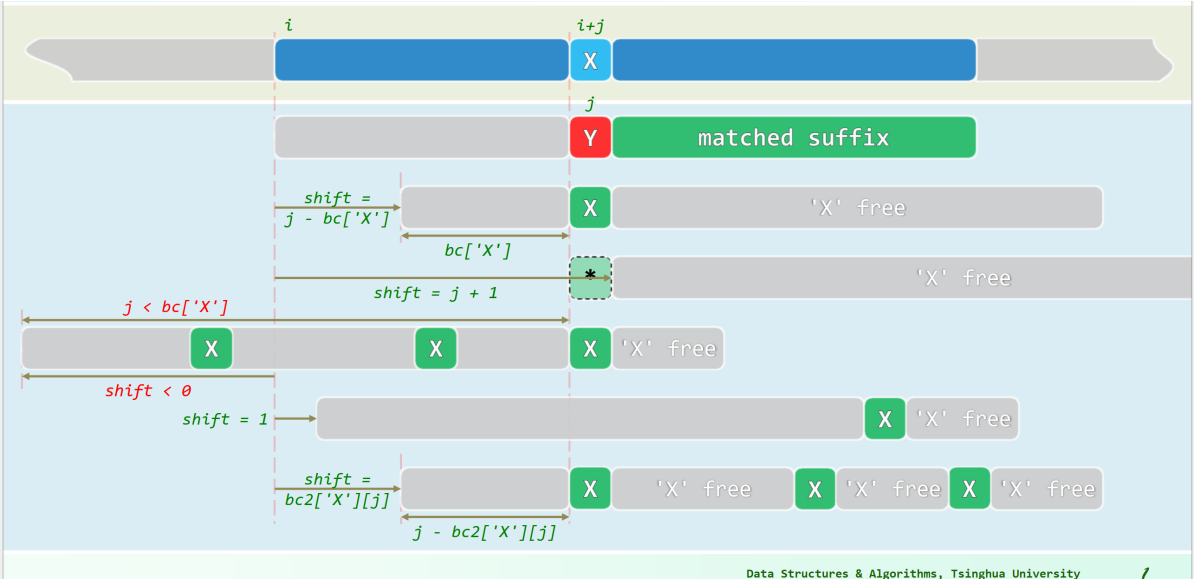
BM算法

R.S.Boyer & J.S.Moore, 1977 ， 方法 2 相对应的。

BC 策略——Bad Character

BC是一个字符集到模式串位置的映射；将每个字符映射到其在字符串最右侧出现的位置。

每次从右到左比较模式串和主串；一旦失配，就将主串的该位置和模式串的BC该字符对齐，随后继续从最右侧开始比较。



时间复杂度

最好情况：$O(n/m)$ 最坏情况 $O(n \times m)$ 。——不够聪明。

GS 策略——Good Sequence

有哪些字符，值得和失配位置主串的字符对齐呢?

image-20210619013558345
image-20210619013623570
image-20210618213054752
image-20210618213109551

构造

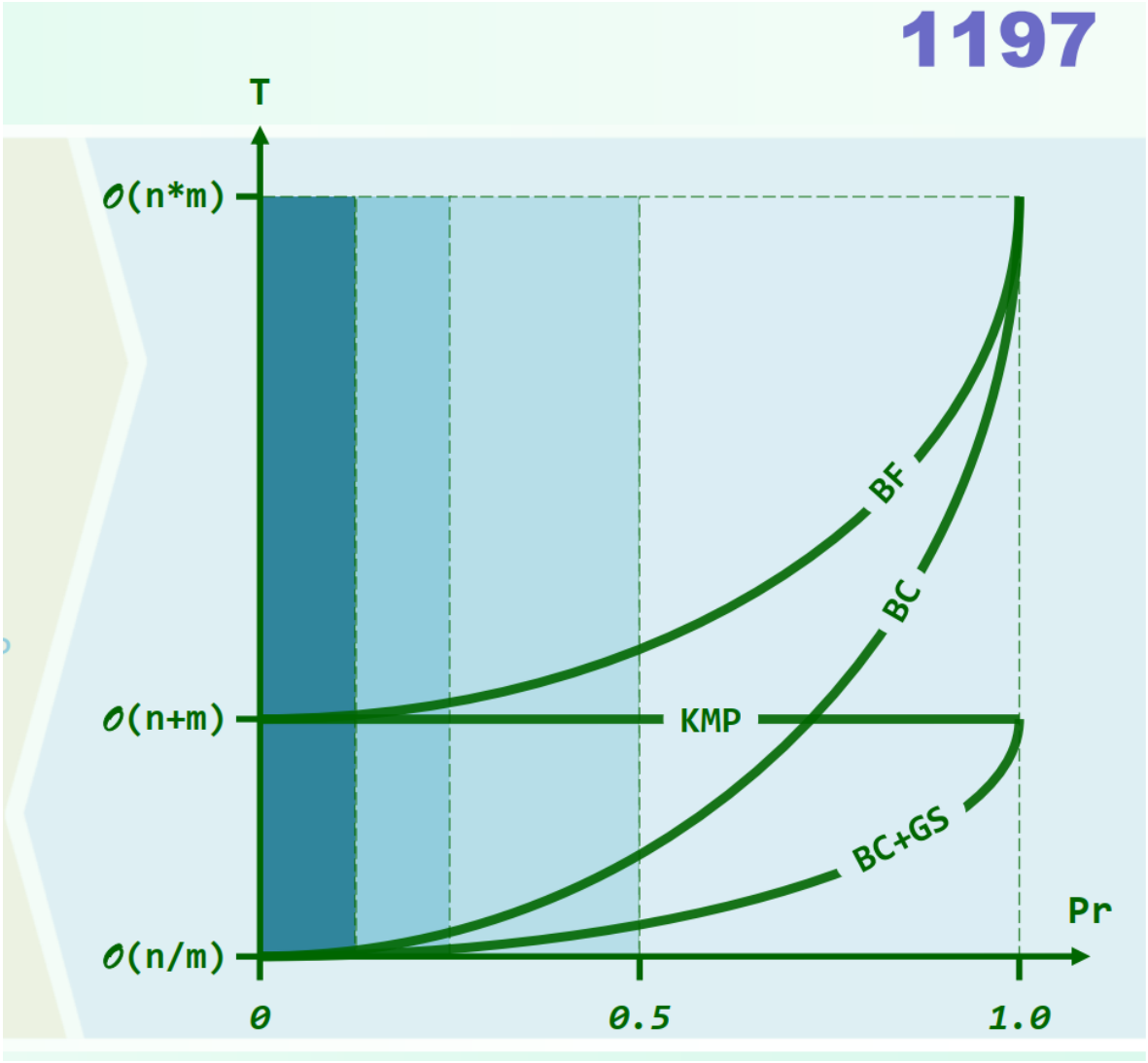
image-20210619013240865
image-20210619013343030
image-20210619013436871
image-20210619013313992
image-20210619013451327

image-20210619013530293

时间复杂度

综合以上两种改进之后，时间复杂度为 $O(n+m)$

几种匹配算法复杂度比较



在真实情况下（字符集比较大），kmp并没有想象中那么好，暴力算法没有想象中那么差

Karp Rabin 算法

散列来比较子串，注意散列筛选只是初筛，最后还需经过严格比对确定最终是否匹配

指纹之间的相关性：以多项式散列函数为例，从上一个指纹用$\mathrm{O}(1)$的时间可以转化为下一个指纹

Trie树

用树结构来存储词，每一个树结点代表一个字母。可以通过二叉树（binary tree）和三叉树（Ternary tree）实现。三叉树的逻辑是只有走中间的单词才作数