

1. Generation Of Test Cases**13B**

本节指出，不能照搬常规的测试方法，简单地用随机的T串、P串，来测试一个串匹配算法的性能。

- 设字符集的规模为 $s = |\Sigma|$ ，如果T串、P串都是随机的，那么匹配成功的概率有多大？
- 对于二进制串 ($s = 2$)，当 $n = 1,000,000$ 、 $m = 100$ 时，这个概率具体是多少？
- 若一台电脑每秒可生成1,000,000,000个这样的测例，需要多久才期望地能得到一个匹配成功的测例？
- 对于一般的字符集，如何随机生成一个匹配成功的测例？

2. Probability Of Worst Cases**13B**

本节给出了Brute-Force算法最坏情况的一个实例，运行时间为 $\mathcal{O}(n \cdot m)$ 。设字符集的规模为 $s = |\Sigma|$ 。

- 在任何一个对齐位置，该算法需要比对 k 次的概率是多少？
- 在任何一个对齐位置，期望地需要做多少次比对？
- 该算法的期望运行时间是多少？

3. Return Rank/Flag**13B**

本节所列的两个版本的Brute-Force算法，返回值分别是“i-j”和“i”。

- 试验证：无论哪个版本，返回值都对应于**最后**那个接受核查的对齐位置；
- 试验证：通过该返回值可以判断出是否找到匹配子串；而且在找到时，该返回值就是子串的位置。

4. Conditional Test**13B**

本节给出的KMP算法中，while循环的条件为“ $(j < m \ \&\& \ i < n)$ ”。不难验证，其中有些对齐的位置（T[i]与P[j]）**不需要**比对即可直接排除。试改进循环条件的表述，精简掉这类对齐位置。

5. For + Else**13B**

本节所列Brute-Force算法的后一版本中，内层for循环有正常终止与提前终止两种情况，且后续的处理也不同。课上指出，尽管这类逻辑模式很常见，但很遗憾C/C++语言并不擅长于描述和实现这类模式。

- 试通过查阅资料，了解Python语言中的“for...break...else...”语法；
- 试以Python语言重写该算法，使得上述逻辑模式更加紧凑，更加易于理解和记忆。

6. Sentinel**13C2**

为更好地理解 and 实现KMP算法，本节建议假想地为每个模式串增设一个**通配符**P[-1]。

- 如果不做如此假想，还有其它什么方式可以简洁地说明、理解和记忆该算法的原理及流程？
- 按照这一假想，如何解释任何模式串必有 $P[0] \equiv -1$ ？

7. Unsuccessful Compares**13C2**

本节在给出KMP算法之后，还绘出了其典型的运行过程，试通过这一可视化的展示确认：

- 算法所做的成功比对次数为 $\mathcal{O}(n)$ ；
- 文本串中的任一字符T[i]，都可能参与**多次**失败的比对；**至多**可能有多少次？

8. N(P, j)**13C3**

本节首先定义了集合N(P, j)，并建议将next[j]取作其中的最大值。

- 试确认，该集合**非空**；

b) 如果 $\text{next}[j]$ 不取作该集合中的最大值, KMP算法需要相应地如何调整?

9. Demos For $\text{next}[]$ Tables

13C[1-4]

试利用老师提供的**演示**工具, 深入理解 $\text{next}[]$ 表的含义、功能及构造过程。

10. Amortization By Aggregate

13C5

本节通过设置一个观察量 " $k = 2*i - j$ ", 简明而严格地证明了KMP算法的线性时间复杂度。然而所谓“知其然更要知其所以然”, 这个观察量具体是何**含义**?

11. Amortization By Accounting

13C5

- a) 试确认, T串中的每一个字符, 至多对应于一次**成功**的比对;
- b) 特别地, 上述结论是否覆盖了与假想的**通配符**所做的“成功”比对?

12. Amortization By Accounting

13C5

本节通过将每次失败的比对分别记账到当时的对齐位置, 同样严格地证明了KMP算法的线性时间复杂度。

- a) 试确认, 每个对齐位置, 确实至多对应于一次**失败**的比对;
- b) 特别地, 在通过假想的**通配符**来对齐时, 如果出现失败的比对, 那么它应该记账至哪个位置?

13. Linear Running Time

13C6

试验证: 即便是本节所举的反例, KMP算法在改进之前**依然**只需运行线性时间。

14. Number Of Compares With Each $T[i]$

13C6

- a) 使用**未改进**的KMP算法, 文本串中的一个字符 $T[i]$, **最多**可能参与多少次比对? 试构造出这样的实例;
- b) 使用**改进后**的KMP算法, 文本串中的一个字符 $T[i]$, **最多**可能参与多少次比对? 试构造出这样的实例。

15. Bad Character

13D2

在根据 $\text{bc}[j]$ 确定的位移量为负数时, 本节建议不妨退至**蛮力**算法, 将P串向前滑动一个单位即可。

- a) 若在此时坚持要确定**最佳**滑动距离, 则需如何调整 $\text{bc}[]$ 表及预处理算法?
- b) 为什么我们倾向于认为, 如此调整得不偿失?

16. Painter's Strategy

13D3

本节给出的 $\text{bc}[]$ 表构造算法, 通过for循环**自前向后**地扫描P串, 每个字符只需 $O(1)$ 时间。如果颠倒过来, 将扫描的方向改为**自后向前**, 会有何不妥?

17. Large Alphabet

13D3

不难发现, 对于较大规模的字符集, $\text{bc}[]$ 中往往会有大量的“-1”。如何消除这类**重复**的信息?

18. Pros & Cons

13D4

试对照本节所给的示意图, 确认各种串匹配算法在不同条件下的性能, 并做对比。

19. Construction Of $\text{gs}[]$

13E2

试按照本节的讲解, 对照示例代码, 理解 $\text{gs}[]$ 表构造算法的正确性, 并验证其线性时间复杂度。