数据结构

夏天

xiat@ruc.edu.cn

中国人民大学信息资源管理学院

字符串

计算机非数值处理的对象经常是字符串数据。如文本检索、文本聚类、文本理解等。

- 串的定义及表示
- 串的模式匹配

串 (String) 的定义

- 串是由零个或多个任意字符组成的字符序列。它也是一种特殊的线性表，其数据元素仅由一个字符组成，而且它通常是作为一个整体来处理。

  例：s="Renmin University"，s 是串名，Renmin University 为串值，其中一个个的字符称为串的元素。
- 请思考如何实现上述串的存储？

字符串的存储

- 顺序存储: 用一组地址连续的存储单元存储串值中的字符序列。
- 链式存储: 考虑到存储密度，可以按块分配。
- 在 C++/Java 等语言中，String 就是字符串。在 C 语言中没有 string 类型变量，字符串用字符数组表示。

串的模式匹配

- 设 s 和 t 是给定的两个串，在主串 s 中寻找等于子串 t 的部分的过程称为模式匹配，t 也称为模式。
- 如果在 s 中找到等于 t 的子串，则称匹配成功，返回 t 在 s 中的首次出现的存储位置，否则匹配失败。
- 例：

  s="ababcabcacbab"

  t="abcac"

简单的模式匹配

s="ababcabcacbab"

t="abcac"

- Brute-Force 算法

BF 算法时间复杂度分析

- 设串 s 长度为 n，串 t 长度为 m。匹配成功的情况下，考虑两种极端情况。
  - 在最好情况下，即每趟不成功的匹配都发生在第一对字符比较时。
  - 例如：s=“aaaaaaaaaabc”，t=“bc”，设匹配成功发生在 s_i 处。
  - 在前面 i-1 趟不成功的匹配中共比较了 i-1 次，第 i 趟成功的匹配共比较了 m 次，所以总共比较了 i-1+m 次。
  - 所有匹配成功的可能共有 n-m+1 种，设从 si 开始与 t 串匹配成功的概率为 p_i，在等概率情况下 $p_i = 1/(n-m+1)$，因此最好情况下平均比较的次数是：

$$\sum_{i=1}^{n-m+1} p_i \times (i-1+m) = \sum_{i=1}^{n-m+1} \frac{1}{n-m+1} \times (i-1+m) = \frac{(n+m)}{2}$$

  - 即最好情况下的时间复杂度是 $O(n+m)$

- 在最坏情况下，即每趟不成功的匹配都发生在 t 的最后一个字符。
- 例如：s=“aaaaaaaaaaab”，t=“aaab”，设匹配成功发生在 s_i 处.
- 在前面 $i-1$ 趟匹配中共比较了 $(i-1)\times m$ 次，第 i 趟成功的匹配共比较了 m 次，所以总共比较了 $i\times m$ 次，因此最坏的情况下平均比较的次数是：

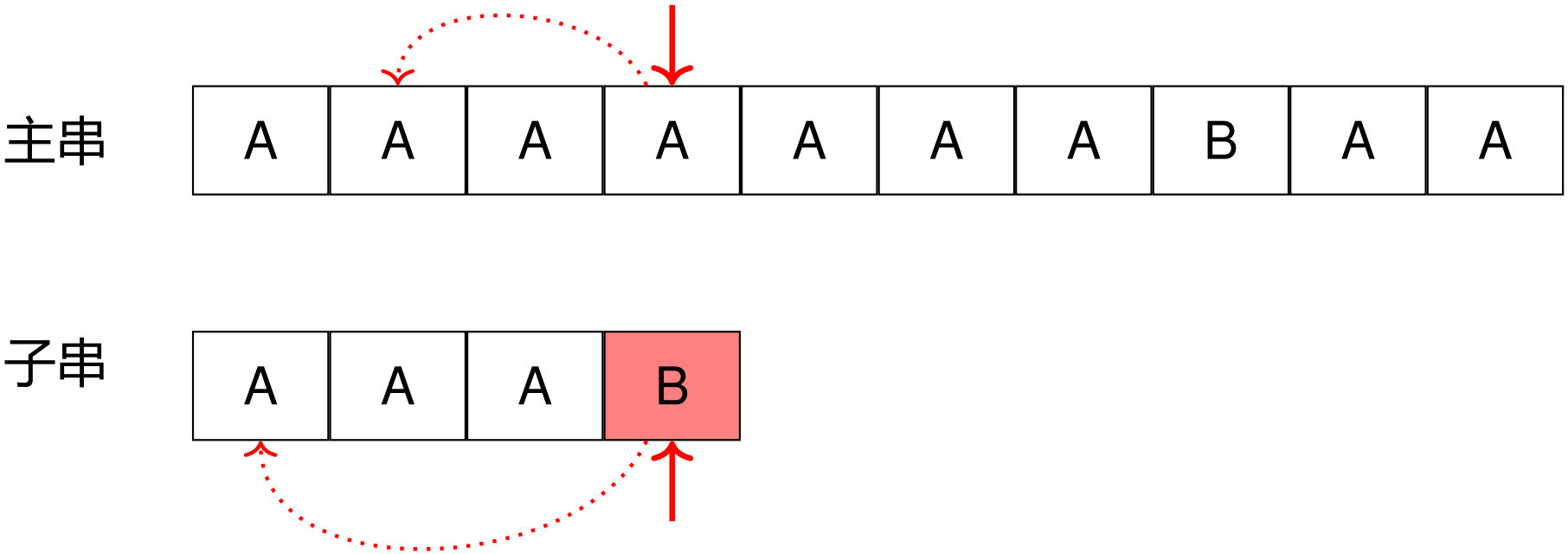
$$\sum_{i=1}^{n-m+1} p_i \times (i \times m) = \sum_{i=1}^{n-m+1} \frac{1}{n-m+1} \times (i \times m) = \frac{m \times (n-m+2)}{2}$$

- 即最坏情况下的时间复杂度是 $O(n\times m)$

为什么 BF 算法时间性能低？

在每趟匹配不成功时存在大量回溯 (每次移动一位开始新的比较)

改进算法



- 改进算法的目的是在每一趟匹配过程中出现不匹配时，向右“滑动”尽可能远的一段距离后，继续进行比较。那么，应当滑动多远呢？这正是各个算法各显神通之处！
- KMP 算法: 由 D. E. Knuth，J. H. Morris 和 V. R. Pratt 同时发现

KMP 算法

利用已经得到的“部分匹配”的结果将模式向右滑动

视频讲解：

https://www.bilibili.com/video/BV1AY4y157yL/

KMP 算法

利用 ChatGPT 来完成代码，已经变得非常方便，参见 string.ipynb 文件。

练习

- 从《红楼梦》中，统计主要人物出现的次数，并按照频次高低输出。
- 文件位置：ipynb/honglou.txt
- 人物列表：ipynb/honglou_person.txt

多模式串匹配与 AC 自动机

- BM、KMP 为单模匹配，即模式串只有一个。假设主串 $\mathrm{T}[1,\cdots,\mathrm{m}]$，模式串有 k 个 $\mathrm{P}=\mathrm{P}_1,\cdots,\mathrm{P}_\mathrm{k}$，且模式串集合的总长度为 n。如果采用 KMP 来匹配多模式串，则算法复杂度为：

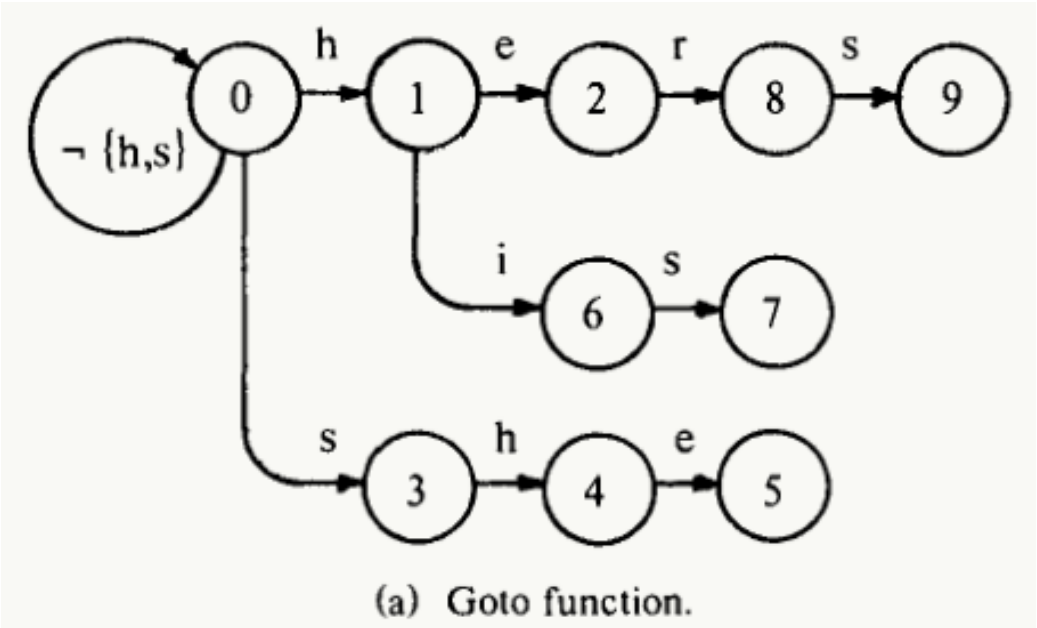
$$\mathrm{O}(|\mathrm{P}_1|+\mathrm{m}+\cdots+|\mathrm{P}_\mathrm{k}|+\mathrm{m})=\mathrm{O}(\mathrm{n}+\mathrm{km})$$

- KMP 并没有利用到模式串之间的重复字符结构信息，每一次模式串的匹配都需要将主串从头至尾扫描一遍。
- 贝尔实验室的 Aho 与 Corasick 于 1975 年基于有限状态机（finite state machines）提出 AC 自动机算法[1]。

[1] AC 算法比 KMP 提出要早，KMP 于 1977 年提出

AC 自动机

- 自动机由状态（数字标记的圆圈）和转换（带标签的箭头）组成，每一次转换对应一个字符。AC 算法的核心包括三个函数：goto、failure、output；这三个函数构成了 AC 自动机。对于模式串 he, his, hers, she，goto 函数表示字符按模式串的转移，暗含了模式串的共同前缀的字符结构信息[2]。



(a) Goto function.

[2] from:https://www.cnblogs.com/en-heng/p/5247903.html

AC 自动机

- failure 函数表示匹配失败时退回的状态：

i	1	2	3	4	5	6	7	8	9
$f(i)$	0	0	0	1	2	0	3	0	3

(b) Failure function.

AC 自动机

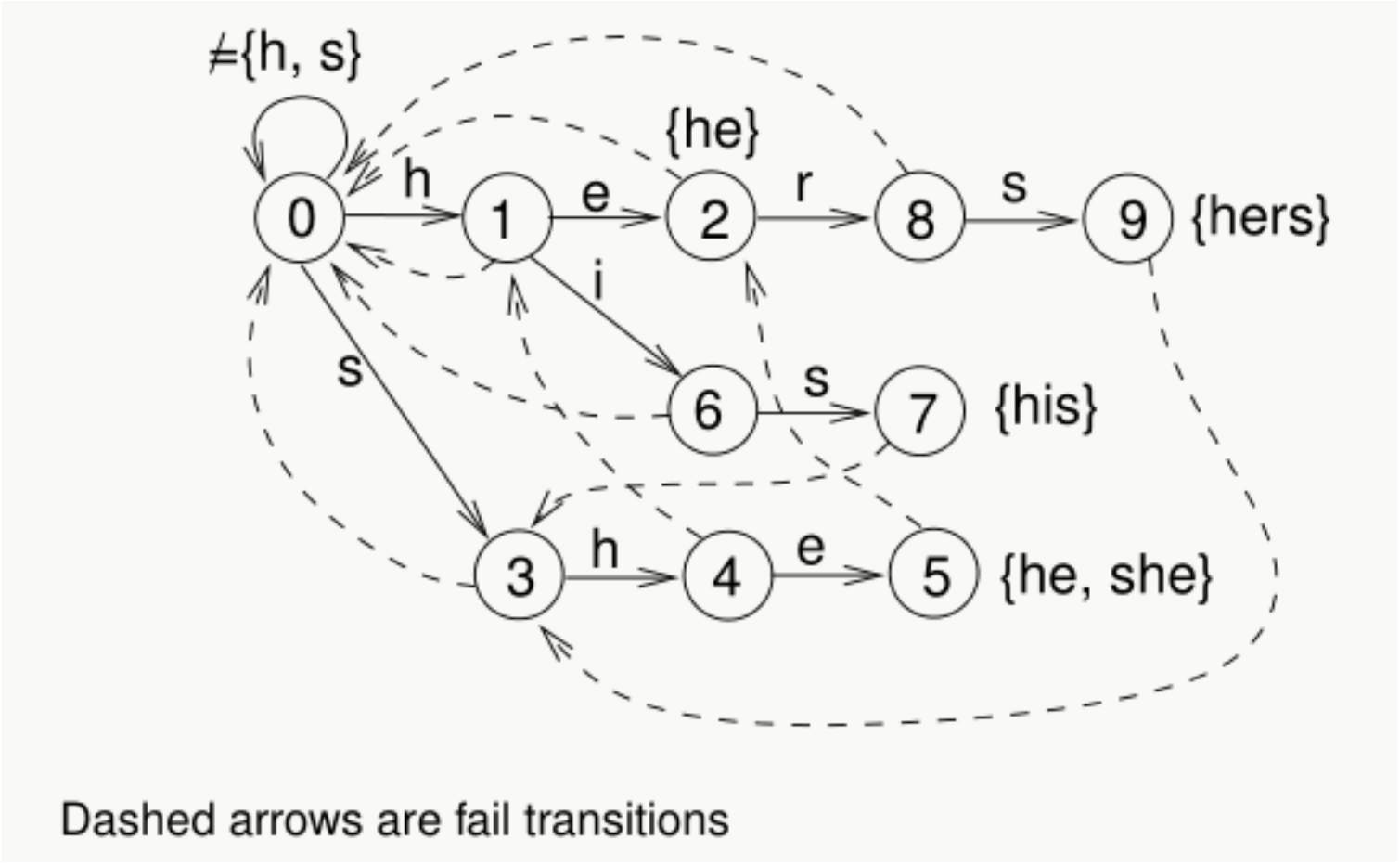
- output 函数表示模式串对应于自动机的状态：

i	$output(i)$
2	{he}
5	{she, he}
7	{his}
9	{hers}

(c) Output function.

AC 自动机

- 完整的 AC 自动机如下：



AC 自动机匹配过程

AC 算法根据自动机匹配模式串，过程比较简单：从主串的首字符、自动机的初始状态 0 开始，

- 若字符匹配成功，则按自动机的 goto 函数转移到下一状态；且若转移的状态对应有 output 函数，则输出已匹配上的模式串；
- 若字符匹配失败，则递归地按自动机的 failure 函数进行转移

Python 实现：pyahocorasick

- https://pypi.org/project/pyahocorasick/
- https://www.bilibili.com/video/BV1yA4y1Z74t/