



第九章 字符串处理算法

张炜
计算机科学与技术学院



Outlines

- 9.1 精确字符串匹配
 - 9.1.1 蛮力算法
 - 9.1.2 指纹算法
 - 9.1.3 基于自动机的算法
 - 9.1.4 KMP算法
 - 9.1.5 BM算法
 - 9.1.6 BMH算法
- 9.2 后缀树及其构造算法
- 9.3 字符串近似匹配及研究前沿



9.1 精确字符串匹配

字符串匹配问题

- 输入:
 - 文本 $T = \text{"at the thought of"}$
 - $n = \text{length}(T) = 17$
 - 模式 $P = \text{"the"}$
 - $m = \text{length}(P) = 3$
- 输出:
 - 所有 s
 - 最小的整数 $(0 \leq s \leq n - m)$ 满足 $T[s..s+m-1] = P[0..m-1]$
 - -1, 如果不存在这样的 s

0 1 2 3 ... $n-1$
 a t t h e t h o u g h t o f
 $s=3$ 0 1 2
 t h e



9.1.1 一个简单直接的算法

基本思想

在 T 的每个位置 i , 检查 P 是否匹配 T 的一个子串

T: ABABABCCA T: ABABABCCA T: ABABABCCA
 P: ABABC P: ABABC P: ABABC

Naive-String-Match(T, P)

Input: 文本 T , 模式 P

Output: P 在 T 中出现的所有位置

1. $n \leftarrow \text{length}(T)$
2. $m \leftarrow \text{length}(P)$
3. For $k \leftarrow 0$ to $n - m$ do
4. IF $P[1..m] = T[k, k+1..k+m-1]$ THEN print k

时间复杂度 $O((n-m+1)m)$



9.1.2 Rabin-Karp算法

- 基本思想
 - $a \equiv b \pmod{n}$ 则 a 可能等于 b
 - 否则, $a \neq b$
 - 将字符串的比较转化成数的比较
- 其性能优于前面的算法
- 易于推广处理其他类似问题
 - 如二维模式匹配问题



0 1 2
 t h e
 20 8 5
 $20 \cdot 27^2 + 8 \cdot 27 + 5 \pmod{19} = 0$ $p=0$

a t t h e t h o u g h t o f
 0 1 2 3 ... $n-1$
 1 20 0 20 8 5 0 20 8 21 15 7 8 20 0 15 6

$t_3 = 20 \cdot 27^2 + 8 \cdot 27 + 5 \pmod{19} = 0$
 $t_2 = 0 \cdot 27^2 + 20 \cdot 27 + 8 \pmod{19} = 16$
 $t_1 = 20 \cdot 27^2 + 0 \cdot 27 + 20 \pmod{19} = 8$
 $t_0 = 1 \cdot 27^2 + 20 \cdot 27 + 0 \pmod{19} = 15$

如何提高计算效率?

HIT CS&E

- $d=|\Sigma|$
 - Σ^* 内任意字符串 x 可以看成是一个 d 进制数
 - 如 $P[0,2,...,m-1]$ 可以看成
 $p = P[m-1] + d(P[m-2] + d(P[m-3] + \dots + dP[0])\dots)$
 计算 p 需要 $O(m)$ 的时间 霍纳法则(Horner's rule)
 - 类似地, $T[k,k+1,...,k+m-1]$ 可以看成
 $t_k = P[k+m-1] + d(P[k+m-2] + d(P[k+m-3] + \dots + dP[k])\dots)$
 计算 t_k 需要 $O(m)$ 的时间
- 如果 $p=t_k$,则 P 在 T 中位置 k 处可能有一个匹配
- 如果在每个位置 k 均重新计算 t_k , 则 $O((n-m+1)m)$
- 能否提高计算 t_k 的效率

HIT CS&E

$$t_k = P[k+m-1] + d(P[k+m-2] + d(P[k+m-3] + \dots + dP[k])\dots)$$

$$t_{k+1} = P[k+m] + d(P[k+m-1] + d(P[k+m-2] + \dots + dP[k+1])\dots)$$

$$t_{k+1} - dt_k = P[k+m] - d^m P[k+1]$$

$$t_{k+1} = dt_k + P[k+m] - d^m P[k+1]$$

– d^m 是与 k 无关的常数, 可以事先计算出来

– 根据 t_k 来计算 t_{k+1} 仅需常数开销

HIT CS&E

0	1	2
t	h	e
20	8	5

$20 \cdot 27^2 + 8 \cdot 27 + 5 \bmod 19 \quad p=0$

a	t	t	h	e	t	h	o	u	g	h	t	o	f
0	1	2	3	...									n-1

1	20	0	20	8	5	0	20	8	21	15	7	8	20	0	15	6
---	----	---	----	---	---	---	----	---	----	----	---	---	----	---	----	---

$t_3 = 20 \cdot 27^2 + 8 \cdot 27 + 5 \bmod 19 = 0 = 27 \cdot t_2 - 27^3 \cdot 0 + 5 \bmod 19$
 $t_2 = 0 \cdot 27^2 + 20 \cdot 27 + 8 \bmod 19 = 16 = 27 \cdot t_1 - 27^3 \cdot 20 + 8 \bmod 19$
 $t_1 = 20 \cdot 27^2 + 0 \cdot 27 + 20 \bmod 19 = 8 = 27 \cdot t_0 - 27^3 \cdot 1 + 20 \bmod 19$
 $t_0 = 1 \cdot 27^2 + 20 \cdot 27 + 0 \bmod 19 = 15$

HIT CS&E

Rabin-Karp-Matcher(T, P, d, q)

Input: 文本 T , 模式 P , 基数 d , 素数 q

Output: P 在 T 中出现的所有位置

1. $n \leftarrow \text{length}(T)$
2. $m \leftarrow \text{length}(P)$
3. $h \leftarrow d^m \bmod q$
4. $p \leftarrow 0$
5. $t_1 \leftarrow 0$
6. For $i \leftarrow 0$ to $m-1$ do //计算 p 和 t_1 $O(m)$
7. $p \leftarrow dp + P[i] \bmod q$
8. $t_1 \leftarrow dt_1 + T[i] \bmod q$
9. for $k \leftarrow 0$ to $n-m$ do $n-m+1$ 遍
10. If $p=t_k$ Then 次数 c
11. If $P[1, \dots, m] = T[k, k+1, \dots, k+m-1]$ Then print k $O(m)$
12. $t_{k+1} \leftarrow dt_k - T[k+1]h + T[k+m] \bmod q$ $O(1)$

HIT CS&E

9.1.3 FA与字符串匹配

i	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	a	b	a	b	a	b	a	c	a	b	a
$P[i]$	a	b	a	b	a	c	a				

扫描 T 的过程

准备扫描

第一个字符	$P[1]$ 已经被匹配
第二个字符	$P[1,2]$ 已经被匹配
第三个字符	$P[1,2,3]$ 已经被匹配
第四个字符	$P[1,2,3,4]$ 已经被匹配
第五个字符	$P[1,2,3,4,5]$ 已经被匹配
第六个字符	$P[1,2,3,4]$ 已经被匹配

.....

HIT CS&E

	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0

i 0 1 2 3 4 5 6 7 8 9 10

$T[i]$ a b a b a b a c a b

$\delta(T[i])$ 0 1 2 3 4 5 6 7 2

Finite-Automation-Matcher(T, δ, m)

Input: 文本 T , FA的状态转移函数, 模式长度 m

Output: P 在 T 中出现的所有位置

1. $n \leftarrow \text{length}(T)$
2. $q \leftarrow 0$
3. For $k \leftarrow 0$ to $n-1$ do
4. $q \leftarrow \delta(q, T[k])$
5. IF $q=m$ THEN print $k-m+1$

如果FA存在, 则找出所有匹配仅需对 $T[1,2,...,n]$ 线性扫描

关键是如何构造这个FA?

模式P的FA的构造

扫描到 $T[j-1]$, 有穷状态机进入状态 $k-1$
 扫描到 $T[j]$ 时

如果 $p_k = t_j$, 则有穷状态机直接进入状态 k
 否则, 假设有穷状态机进入状态 r , 意味着...
 $t_1, t_2, \dots, t_{j-1}, t_j$ 的所有后缀中能够与 P 的某个前缀匹配的最大长度等于 r
 $\forall x \in \Sigma$ 令 $\sigma(x) = \max\{k: P_k \triangleright x\}$ 且 $\sigma(\epsilon) = 0$
 例如 $P = ab$ 则 $\sigma(\epsilon) = 0$ $\sigma(ccaca) = 1$ $\sigma(ccab) = 2$
 则 $\delta(k-1, a) = \sigma(T_{j-1}a) = \sigma(P_{k-1}a)$

Compute-Transition-Function(P, Σ)

Input: 模式 P , 字符表 Σ
 Output: P 对应有穷自动机的状态转移函数 δ

1. $m \leftarrow \text{length}(P)$
2. For $q \leftarrow 0$ to m do
3. For $\forall a \in \Sigma$ do
4. $k \leftarrow \min\{m+1, q+2\}$
5. repeat $k \leftarrow k-1$ until $P_k \triangleright P_q a$
6. $\delta(q, a) \leftarrow k$
7. return δ ;

时间复杂度 $O(m^3|\Sigma|)$
 可以用更快的算法将时间复杂度改进到 $O(m|\Sigma|)$

9.1.4 Knuth-Morris-Pratt算法

考察最简单最直接的扫描过程

$T = \text{b a c b a b a b a b c b a b}$
 $P = \text{a b a b a c a}$

$-P_5$ 已经被匹配上但 P_6 不匹配
 $-$ 由于 $a \neq b$, 在 $s+1$ 开始的扫描不可能得到正确匹配, 应跳过
 $-$ 下一个可能正确的匹配应起始于何处?

$T = \text{b a c b a b a b a b c b a b}$
 $s' = s+2 \rightarrow \text{a b a b a c a}$

一般而言, 下面的问题可以加速扫描过程
 设 $P[1, \dots, q]$ 匹配了 $T[s+1, \dots, s+q]$, 最小的 s' 是什么, 使得
 (1) $s' > s$; (2) $P[1, \dots, k] = T[s'+1, \dots, s'+k]$; (3) $s+q = s'+k$

$T = \text{b a c b a b a b a b c b a b}$
 $P_q = \text{a b a b a}$
 $P_k = \text{a b a}$

只需将 P_q 与其自身进行比较
 等价于找最大的 k 使得 $P_k \triangleright P_q$
 无需存储 s' , 只需存储 $s'-s$

如果上述问题知道答案
 $-$ 如果 $T[s+q+1] = P[q+1]$ 扫描继续进行
 $-$ 否则

- 扫描跳过若干无效扫描位置
- 直接转入起始于 $s'+1$ 开始新的扫描
- 在新的扫描位置, 有 k 个字符已匹配, 无需再扫描

答案要借助哪些信息才能计算, 如何计算?

前缀函数—可用于高效计算9.1.3中的状态转移函数

给定模式 $P = P[1, 2, 3, \dots, m]$, 模式 P 的前缀函数是一个函数 $\pi: \{1, 2, \dots, m\} \rightarrow \{0, 1, 2, \dots, m-1\}$ 使得 $\pi(q) = \max\{k: k < q \text{ 且 } P_k \triangleright P_q\}$

i 1 2 3 4 5 6 7 8 9 10
 $P[i]$ a b a b a b a b c a
 $\pi[i]$ 0 0 1 2 3 4

$\pi(1)=0$
 $\pi(2)=0$
 $\pi(3)=1$
 $\pi(4)=2$
 $\pi(5)=3$
 $\pi(q)=k+1$ 上述过程可以重复, 最坏情况下直到 $k=0$
 $\pi(6)=4$ 从而避免两重循环的扫描
 经过平摊分析知上述过程的时间开销为 $O(m)$

$P_5 = \text{a b a b a}$
 $P_6 = \text{a b a b a b}$
 $P_q = \text{a b a b a}$ $k=3$

Compute-Prefix-Function(P)

Input: 模式 P
 Output: P 的前缀函数 π

1. $m \leftarrow \text{length}(P)$
2. $\pi[1] \leftarrow 0$; $k \leftarrow 0$;
3. For $q \leftarrow 2$ to m do
4. While $k > 0$ & $P[k+1] \neq P[q]$ do
5. $k \leftarrow \pi[k]$;
6. If $P[k+1] = P[q]$ then $k \leftarrow k+1$;
7. $\pi[q] \leftarrow k$;
8. return π ;

第5步/每次, 势能增量 ≤ -1
 第6步/每次, 势能增量 1

For 循环体的每次执行
 \triangleright 第5步执行遍数 x
 \triangleright While 循环代价为 x
 \triangleright 第6步执行 1 遍
 \triangleright 第6步的代价为 1

For 循环体的每次执行
 $\triangleright \alpha = c + \text{势能增量}$
 $= (x+1) + \text{势能增量}$
 ≤ 1 ($x \neq 0$)
 ≤ 2 ($x = 0$)

令状态 k 的势能为 k
 经过平摊分析知上述过程的时间开销为 $O(m)$

KMP-Matcher(P, T)

Input: 模式 P, T

Output: P 在 T 匹配的所有起始位置


1. $m \leftarrow \text{length}(P); n \leftarrow \text{length}(T)$
2. $\pi \leftarrow \text{Compute_Prefex_Function}(P);$
3. $q \leftarrow 0;$ //用于跟踪已匹配的字符数
4. For $i \leftarrow 2$ to n do //从左到右扫描 T
5. While $k > 0$ & $P[q+1] \neq T[i]$ do
6. $q \leftarrow \pi[q];$
7. If $P[q+1] = T[i]$ then $q \leftarrow q+1;$
8. If $q = m$ then
9. print $i-m+1;$
10. $q \leftarrow \pi[q];$

令状态 q 的势能为 q , 经过平摊分析知上述过程的时间开销为 $O(m+n)$

9.1.5 Boyer-Moore算法

HIT CS&E

- 还存在比KMP算法效率更高的匹配算法吗?
- 还能从其他角度处理字符串匹配问题吗?
- 常用的字符串匹配算法是如何实现的呢?



- 德克萨斯大学的Robert S. Boyer和J Strother Moore 1977年发明的算法

基本思想

HIT CS&E

将输入串 T 与模式 P 的开始位置对齐, 从后往前比较

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

模式 P 的最后一个字符没匹配上, 意味着...

- “坏”字符 - 刻画了 P 在整体上与 T 在该位置的不相似
 - T, P 位置对齐
 - 从后向前扫描
 - 首个不匹配字符
- P 需要向后移动若干个字符才可能发生匹配

$T[6] \neq P[6]$ 且 $T[6] \notin P$, 意味着...

- P 至少向后移动 $|P|$ 个位置才可能发生匹配

基本思想 (续)

HIT CS&E

将输入串 T 与模式 P 的开始位置对齐, 从后往前比较

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

$T[13] \neq P[6]$ 但 $T[13] = P[4]$, 意味着...

- P 至少向后移动 $6-4=2$ 个位置可能发生匹配

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

基本思想 (续)

HIT CS&E

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

好后缀 : $P[6]$ 出现位置: 6

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

好后缀 : $P[5..6]$ 出现位置: 5

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

好后缀 : $P[4..6]$ 出现位置: 4

“好”后缀: 从后向前扫描“坏字符”前遇到每个 P -后缀

基本思想 (续)

HIT CS&E

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

好后缀 : $P[3..6]$ 出现位置: 3

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
EXAMPLE
0 1 2 3 4 5 6
  
```

“坏”字符又出现了, 应该把模式向后移动几个位置?

“坏”字符规则

- 如果 $T[i] \neq P[j]$ 且, 则后移次数 $k = j - a$
- $a = T[i]$ 最近扫描时在 $P[j]$ 前最右的出现位置
- 如果 $P[j]$ 前没有出现过 $T[i]$, 则 $a = -1$

HIT CS&E

基本思想 (续)

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

HERE IS A SIMPLE EXAMPLE

EXAMPLE

除了“坏”字符规则外，还有其他机制使移动更高效吗？

HIT CS&E

基本思想 (续)

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

“坏”字符又出现了，应该把模式向后移动几个位置？

“好”后缀规则

- 如果 $T[i] \neq P[i]$ 且，则后移次数 $k = b - a$
- b = 最近扫描时“好”后缀在 P 中的出现位置
- a = 好后缀在模式串中上一次出现的位置，且如果好后缀在模式串中没有再次出现，则为 -1

HIT CS&E

基本思想 (续)

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

“坏”字符又出现了，应该把模式向后移动几个位置？

“坏”字符规则，后移次数 = $2 - (-1) = 3$

“好”后缀规则，后移次数 = $6 - (0) = 6$

HIT CS&E

基本思想 (续)

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

最终规则

- 后移次数 $k = \max\{\text{好后缀规则}, \text{坏字符规则}\}$
- “坏”字符规则 = $6 - 4 = 2$
- “好”后缀规则 = $0 - (-1) = 1$

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

哪些信息可以预计算以提高效率？

算法如何形式化描述？

EXAMPLE
 0 1 2 3 4 5 6

算法的复杂性如何？

HIT CS&E

9.1.6 Boyer-Moore-Horspool算法

- 还存在比BM算法效率更高的匹配算法吗？
 - BM算法仍存在冗余扫描
 - 跨度时机不好选择
- 德克萨斯大学的Robert S. Boyer和J Strother Moore
 - 1977年发明的算法
 - 不再关注失配字符
 - 而是关注失配发生时，与P对齐的最后一个字符 x

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

后移量 = $|P| - 1 - x$ 在 P 中的最后出现位置


后移量 = $|P|$ x 在 P 中未出现

HIT CS&E

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 HERE IS A SIMPLE EXAMPLE

EXAMPLE
 0 1 2 3 4 5 6

后移量 = $7 - 1 - 4 = 2$




HIT
CS&E

基本思想（续）

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6

```



HIT
CS&E


基本思想（续）

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6

```

后移量 = 7 I未在P中出现



HIT
CS&E

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6

```


后移量 = 7-1-5=1 L在P[5]出现

```

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
HERE IS A SIMPLE EXAMPLE
          EXAMPLE
            0 1 2 3 4 5 6

```

理解BMH算法之后，写出代码并做分析



HIT
CS&E

9.2 后缀树及其构造算法


- 在模式P给定且被预处理的情况下，任给文本T, 模式匹配问题可以在 $O(|T|)$ 时间内求解
 - ***信息过滤系统
 - 在病人DNA序列中寻找给定疾病的DNA模式
- 在文本T给定且可以被预处理的情况下，任给模式P, 模式匹配问题需要多长时间才能求解？怎样求解？
 - 静态网页中的关键字搜索
 - 数字图书馆中的信息查询

这些实际问题中的科学问题到底是什么？



HIT
CS&E

- 预处理 P
 - Gusfield
 - Boyer-Moore
 - Knuth-Morris-Pratt
 - Boyer-Moore-Horspool
- 预处理 T
 - 后缀树



HIT
CS&E

9.2.1 相关概念

- 问题定义
 - Input: 模式P 和文本 S
(S可被预处理, T有其他用途)
 - Output: P在S中的一次出现
- 例
 - S = b b a b b a a b
 - P = b a a

HIT CS&E

后缀

$T = \text{b b a b b a a b}$

$T[1...8] = \text{b b a b b a a b}$	1-后缀
$T[2...8] = \text{b a b b a a b}$	2-后缀
$T[3...8] = \text{a b b a a b}$	3-后缀
$T[4...8] = \text{b b a a b}$	4-后缀
$T[5...8] = \text{b a a b}$	5-后缀
$T[6...8] = \text{a a b}$	6-后缀
$T[7...8] = \text{a b}$	7-后缀
$T[8...8] = \text{b}$	8-后缀

注意: P 在 S 中出现当且仅当 P 是某个 i -后缀的前缀
 所谓预处理就是恰当地组织左右后缀, 使得...

HIT CS&E

$T = S$ 的 suffix trie

下列论断等价

- P 在 S 中出现
- P 是 S 的某个后缀的前缀.
- P 在 S 的后缀树 T 中对应一条起始于根的路径.

HIT CS&E

$P = \text{b a b b a}$

P 在 S 中出现!

HIT CS&E

$P = \text{b b a a b a}$

P 未出现在 S 中!

HIT CS&E

$P = \text{a b b b a a}$

P 未出现在 S 中!

HIT CS&E

$P = \text{a b b a a}$

输出3!



问题

存储suffix trie的空间开销为

$\Theta(|S|)$ $\Theta(|S| \log |S|)$ $\Theta(|S|^2)$ $\Theta(|S|^3)$

为S构建其suffix trie T的时间开销为多少? $\Theta(|S|^2)$

是否存在suffix trie的更紧凑的表示使得其
构建时间和存储空间均是线性的?



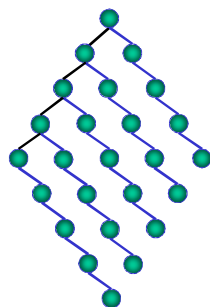
后缀树

suffix trie的一种更紧凑的表示



观察S的suffix Trie T

- T至多有 $|S|$ 个叶子.
– Why?
- T至多有 $|S|$ 个分叉内节点.
– Why?

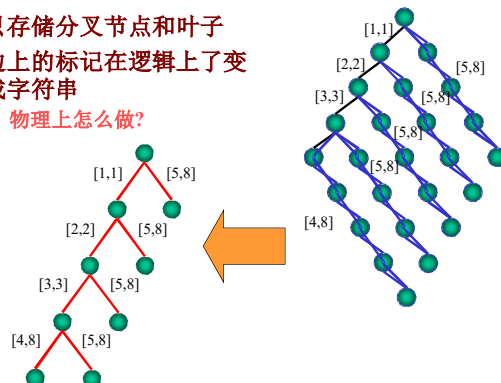


后缀树

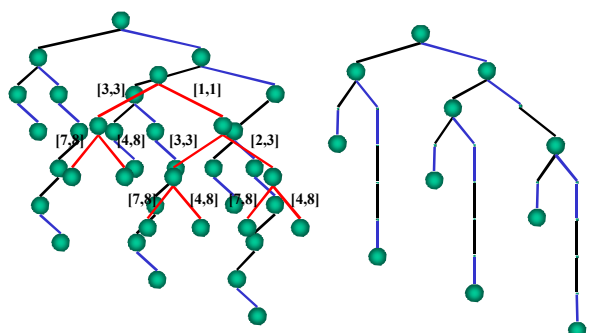


S = a a a a b b b b

- 只存储分叉节点和叶子
- 边上的标记在逻辑上变成了字符串
– 物理上怎么做?



S = b b a b b a a b



问题

- 后缀树的空间开销
 - $O(|S|)$
 - $O(|S| \log |S|)$
 - $O(|S|^2)$
 - $O(|S|^3)$
- 为什么?
 - 后缀树中的节点数 =
 - 后缀树中的边数 =
 - 每条边需要的存储空间 =

挑战

如何在线性时间内构造后缀树?



HIT
CS&E

后缀树的历史点滴

- [Weiner, *IEEE FOCS* 1973]
 - 线性时间 空间开销巨大.
 - D. E. Knuth: “the algorithm of 1973”.
- [McCreight, *J. ACM* 1976]
 - $O(n)$ 时间 $O(n^2)$ 空间
- [Ukkonen, *Algorithmica* 1995]
 - $O(n)$ 时间 $O(n)$ 空间
 - 更稳定可靠.




HIT
CS&E



Academy Professor ,
Department of Computer
Science , University of
Helsinki, Finland
<http://www.cs.helsinki.fi/u/ukkonen/>

Esko Ukkonen: On-line construction of
suffix-trees. *Algorithmica* 14 (1995), 249-
260

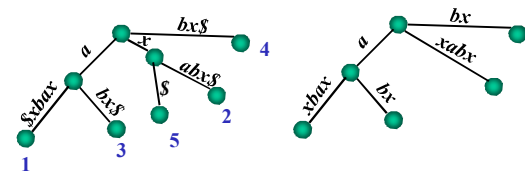



HIT
CS&E

隐式后缀树

SS 的后缀树 T 中

- step 1. 删除 S 符号
- step 2. 删除无标签的边
- step 3. 删除只有一个孩子的顶点






HIT
CS&E

算法基本思想

1. T_1 为 $S[1]$ 的隐式后缀树
2. For $i=2$ to $|S|$
3. 更新 T_{i-1} 为 $S[1, \dots, i]$ 的隐式后缀树 T_i
4. end for
5. 更新 $T_{|S|}$ 为 SS 的隐式后缀树, 即 S 的后缀树

关键: 如何加速第3步



HIT
CS&E


Ukkonen's构建后缀树的方法

$S = b b a b b a a b$

$T[0]$
 $S[1] = b$
 $T[0]$ 经过什么变化才能变为 $T[1]$?

$T[1]$
 经过一次叶柄变长
 $S[2] = b$
 $T[1]$ 经过什么变化才能变为 $T[2]$?

$T[2]$
 经过一次叶柄变长
 经过一次若无其事
 $S[3] = a$
 $T[2]$ 经过什么变化才能变为 $T[3]$?



HIT
CS&E

Ukkonen's构建后缀树的方法

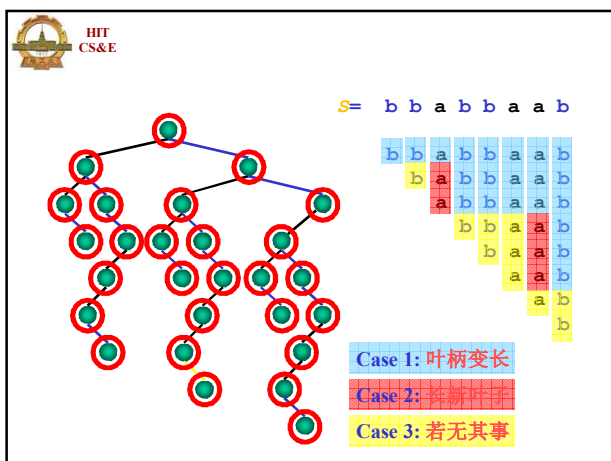
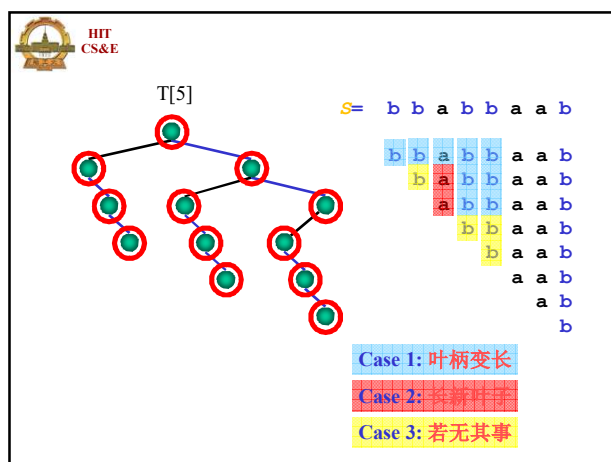
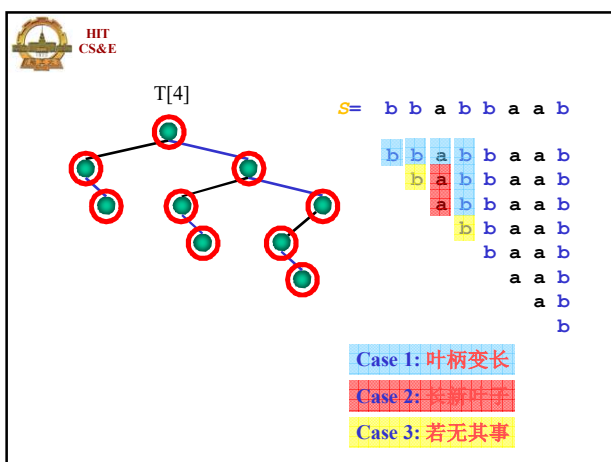
$S = b b a b b a a b$

$T[3]$

经过一次叶柄变长
 经过两次长新叶子
 $S[4] = b$
 $T[3]$ 经过什么变化才能变为 $T[4]$?

Case 1: 叶柄变长
 Case 2: 长新叶子
 Case 3: 若无其事

9



构建后缀树时涉及到三类动作

- Case 1: 在一个叶子节点上延申边，叶柄变长
- Case 2: 在一个内节点处长新叶子，长新叶子
- Case 3: 树结构不发生任何变化，若无其事

三阶段定理

处理 $S[k]$ 的 k 个步骤由如下一系列步骤构成：
(或，处理后缀的第 k 列的 k 个步骤如下构成：)

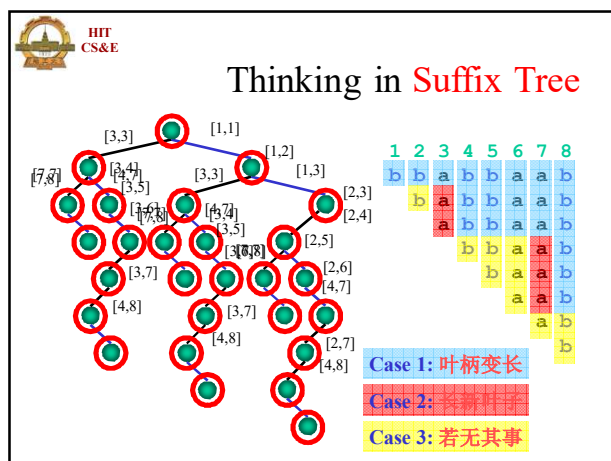
- 首先是一系列(至少一个)叶柄变长
- 然后是一系列(可能0个)长新叶子
- 最后是一系列(可能0个)若无其事

引理1. 在构建 T_1, \dots, T_m 的过程中，一旦一个叶子产生出来就不会消失，后续过程中该叶子可能的变化只能是叶柄变长

引理2. 将构建 T_i 的过程中最后一次长新叶子的操作时刻记为 j_i ，则 $j_i \leq j_{i+1}$

引理3. 在构建 T_i 的过程中第一次若无其事发生后，以后的操作均是若无其事

结合下面的实例，自己给出证明



利用三阶段定理加速构建

- 利用形如 $[i,-]$ 标记叶边, $-$ 是记录字符串结束位置的标记, **叶柄变长**无需显式操作, 常数时间即可完成所有**已知**的叶柄变长
- J_i 记录 T_i 构建过程中最后一次长新叶子的操作位置, 帮助我们定位已知的叶子

利用三阶段定理加速构建

i	1	2	3	4	5	6	7	8
$J_1=1$	b	a	b	b	a	a	b	
$J_3=3$	b	a	b	b	a	a	b	
$J_3=3$	a	b	b	a	a	b		
$J_4=3$	b	b	a	a	b			
$J_5=3$	b	a	a	b				
$J_7=6$	a	a	b					
$J_8=6$	a	b						

Case 1: 叶柄变长

Case 2: 新增叶子

Case 3: 若无其事

进一步加速—思想梗概

$T_i > T_{i+1}$

考虑将 $S[j, \dots, i]S(i+1)$ 添加到 T_i 的过程

- $S[j, \dots, i]$ 必然被某条路径标记
- 结束位置不一定是结点
- 如何有效发现距结束位置最近的结点

将 $S[j-1, j, \dots, i]S(i+1)$ 添加到 T_i 时

- 经过路径 $S[j-1, j, \dots, i]S(i+1)$
- 中间某个结点有指针指向 α 的结束结点
- 省略很多不必要的路径访问
- 这种指针被称作**suffix link**

Suffix link

T_i 中从根结点出发到结点 u 的路径标记为 $x\alpha$, 则结点 u 的**suffix link**指向结点 v , 其中 x 是一个字符且从根结点到 v 的路径标记为 α

Suffix Links性质

性质1

- suffix link**的出发点总是后缀树的内部节点
 - 不可能是叶节点
 - 也不可能是一条边的中间位置
- 为什么?

性质2

- 后缀树的每个内部节点均是某个**suffix link**的出发点
- 为什么?

性质2的证明

设

- T_i 已经构造出来
- 每个内结点均有一个**suffix link**
- $J_i = k \geq 1$

$T_i > T_{i+1}$

- $S[1, 2, \dots, i]S(i+1) \rightarrow T_i$
- $S[2, \dots, i]S(i+1) \rightarrow T_i$
- ...
- $S[k, \dots, i]S(i+1) \rightarrow T_i$

(4) 上述操作均是叶柄变长—隐式地完成
不增加新的内结点, 故各内结点均有**suffix link**

(5) 只需考虑 $S[j, \dots, i]S(i+1) \rightarrow T_i$ ($j \geq k \geq 2$)

HIT CS&E

性质2的证明(续)

$T_i \rightarrow T_{i+1} : S[j, \dots, i] S(i+1) \rightarrow T_i \ (j \geq k \geq 2)$

- (1) $S[j-1, j, \dots, i, i+1]$ 在当前 T_{i+1} 对应叶子
- (2) 从该叶子上行找到第一个有 suffix link 的顶点 v
- (3) 经过路径的标签为 γ
- (4) 由 suffix link 从 v 到达 $s(v)$
- (5) 从 $s(v)$ 沿标记为 γ 的路径前进，或长新叶子，或若无其事

问题：如果第(5)步添加内结点 w 从而长新叶子， $s(w)$ 从何而来？

HIT CS&E

Suffix link的维护

$T_i \rightarrow T_{i+1} : S[j, \dots, i] S(i+1) \rightarrow T_i \ (j \geq k \geq 2)$

添加了内结点 w

$S[j+1, \dots, i] S(i+1) \rightarrow T_i$ 添加过程相同

从 $s(v)$ 沿标记为 δ 的路径前进，在 δ 结束位置有两种情况

Case 1: δ 结束于某个结点 u
 $s(w) = u$

Case 2: δ 结束于某条边内部
 $S(i+1)$ 必然导致新的内结点 u 产生
 $s(w) = u$

HIT CS&E

运用后缀链加速后缀树的构建

1 2 3 4 5 6 7 8
 $S = b b a b b a a b$
 $S_1 = b$
 将 S_1 的所有后缀插入 T
 $J_1 = 1$

HIT CS&E

运用后缀链加速后缀树的构建

1 2 3 4 5 6 7 8
 $S = b b a b b a a b$
 $S_2 = b \underline{b}$
 将 S_2 的所有后缀插入 T
 $J_2 = 1$

HIT CS&E

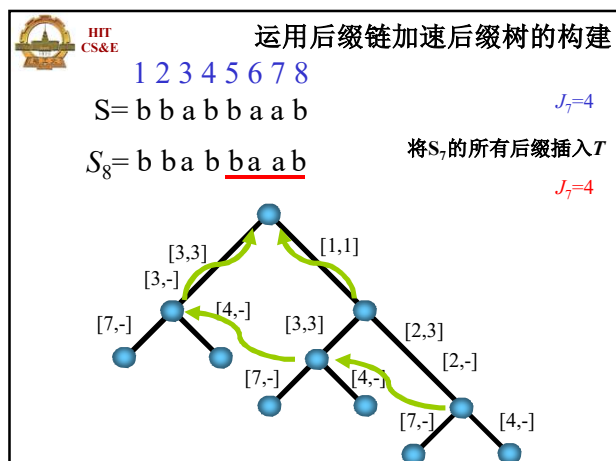
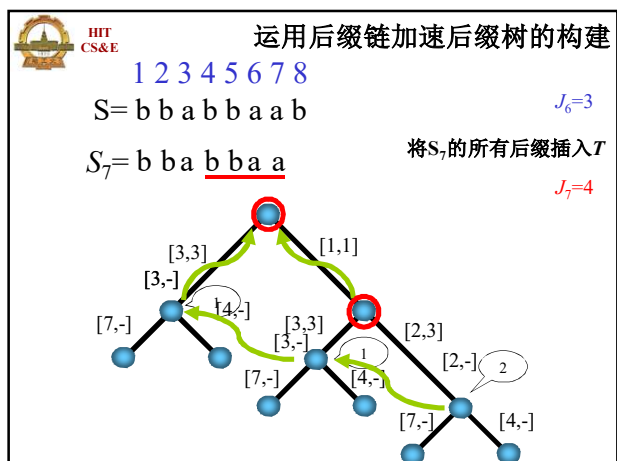
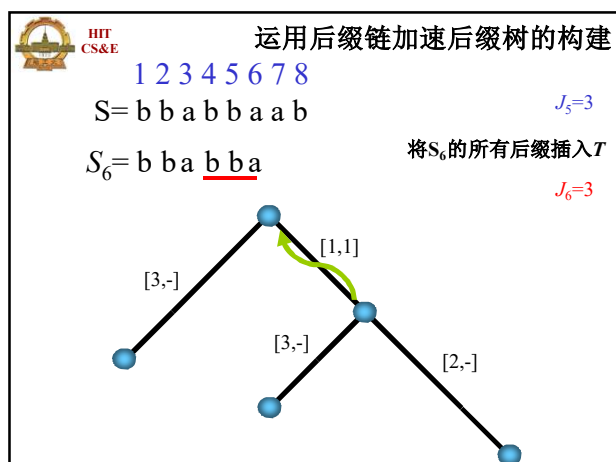
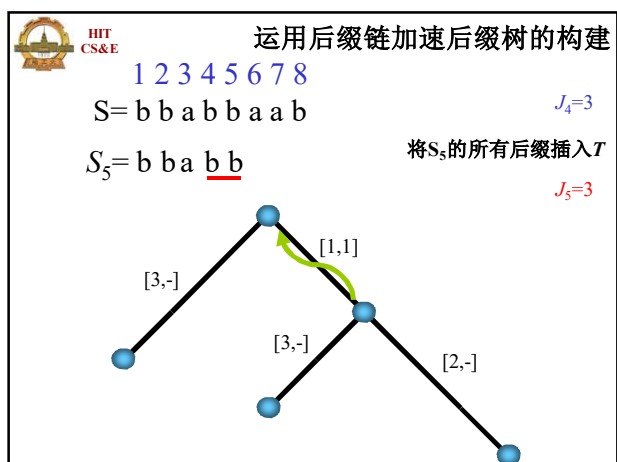
运用后缀链加速后缀树的构建


1 2 3 4 5 6 7 8
 $S = b b a b b a a b$
 $S_3 = b \underline{b} \underline{a}$
 将 S_3 的所有后缀插入 T
 $J_3 = 3$

HIT CS&E

运用后缀链加速后缀树的构建

1 2 3 4 5 6 7 8
 $S = b b a b b a a b$
 $S_4 = b b a \underline{b}$
 将 S_4 的所有后缀插入 T
 $J_4 = 3$






HIT
CS&E

Suffix tree的构造时间

- $O(n)$ —略



9.3 近似字符串匹配

- 问题定义
- 相似性度量函数
- 一些典型求解思路及研究前沿

HIT CS&E

DBLP 作者搜索

HIT CS&E

尝试一下这些名字 (good luck!)



UCSD

Yannis Papakonstantinou



Case Western

Meral Ozsoyoglu



AT&T--Research

Marios Hadjieleftheriou

<http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/index.html>

HIT CS&E

☹

HIT CS&E

更好的系统?

HIT CS&E

UC Irvine 的人名搜索

HIT CS&E

Web Search

Google 搜集的实际查询

- 查询的错误
- 数据的错误
- 使得查询和有意义结果接近

<http://www.google.com/jobs/britney.html>

HIT CS&E

数据清洗

R

informix
microsoft
...
...

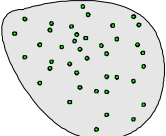
S

infromix
...
mcrosoft
...

85

HIT CS&E

问题的定义



输入: 字符串 q , 字符串集合 D , 相似性函数 $\text{dist}(.,.)$
 输出: D 中与 q 相似的所有字符串: $\text{dist}(Q,D) \leq \delta$
 例: 找到和“**had**j**ele**f**theriou**”相似的字符串

你有什么办法呢?

性能很重要!

- 10 ms: 100 查询/秒 queries per second (QPS)
- 5 ms: 200 QPS

86

HIT CS&E

基础知识

87

HIT CS&E

相似性函数

- 相似性函数:
 - 领域相关函数
 - 返回字符串间的相似性值
- 例如:
 - 编辑距离
 - Hamming距离
 - Jaccard相似性
 - Soundex
 - TF/IDF, BM25, DICE
 -

88

HIT CS&E

编辑距离

- 一种广泛使用的字符串相似性测度
- $\text{Ed}(s1, s2)$ = 将 $s1$ 变化到 $s2$ 需要的最小操作数
 - 增加、删除、修改
- 例:
 - $s1$: Tom Hanks
 - $s2$: Ton Hank
 - $\text{ed}(s1, s2) = 2$

89

HIT CS&E

技术: Oracle 10g

- Oracle Text
- `CREATE TABLE engdict(word VARCHAR(20), len INT);`
- 创建文本索引:


```
begin ctx_ddl.create_preference('STEM_FUZZY_PREF', 'BASIC_WORDLIST');
ctx_ddl.set_attribute('STEM_FUZZY_PREF', 'FUZZY_MATCH', 'ENGLISH');
ctx_ddl.set_attribute('STEM_FUZZY_PREF', 'FUZZY_SCORE', '0');
ctx_ddl.set_attribute('STEM_FUZZY_PREF', 'FUZZY_NUMRESULTS', '5000');
ctx_ddl.set_attribute('STEM_FUZZY_PREF', 'SUBSTRING_INDEX', 'TRUE');
ctx_ddl.set_attribute('STEM_FUZZY_PREF', 'STEMMER', 'ENGLISH'); end;
```
- `CREATE INDEX fuzzy_stem_subst_idx ON engdict (word) INDEXTYPE IS ctxsys.context PARAMETERS ('Wordlist STEM_FUZZY_PREF');`
- 使用方法:


```
SELECT* FROMengdict
WHERE CONTAINS(word, 'fuzzy(university, 70, 6, weightY, 1)') > 0;
```
- 限制: 不能处理首字母错误:
 - Katherine VS Catherine

90



Microsoft SQL Server

- SQL Server 2005提供数据清洗工具
- 信息集成工具的一部分
- 支持模糊查询
- 相似性函数: 基于TF/IDF的打分

91



Lucene

- 使用Levenshtein Distance (编辑距离).
- 先基于前缀过滤, 然后执行搜索 (效率?)

92



基于Gram的算法

93



"q-grams"

u n i v e r s a l

2-grams

94



编辑操作和 gram的关系

u n i v e r s a l

固定长度: q

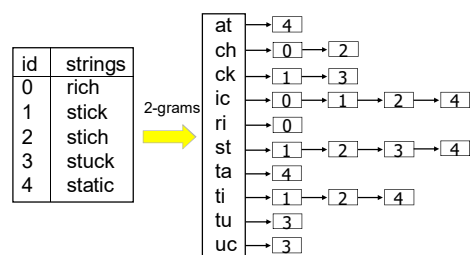
k 个操作会影响 $k * q$ 个 gram

如果 $ed(s_1, s_2) \leq k$, 那么他们公共gram的数量 \geq
 $(|s_1| - q + 1) - k * q$

95



q-gram 的倒排表



96

使用倒排表的搜索

- 查询: "shtick", $ED(shtick, ?) \leq 1$
 sh ht ti ic ck 公共 gram 的数量 ≥ 3

id	strings
0	rich
1	stick
2	stich
3	stuck
4	static

2-grams

```

at → 4
ch → 0 → 2
ck → 1 → 3
ic → 0 → 1 → 2 → 4
ri → 0
st → 1 → 2 → 3 → 4
ta → 4
ti → 1 → 2 → 4
tu → 3
uc → 3
    
```

T 出现问题

合并

递增顺序

查找出现次数 $\geq T$ 的元素

例子

- $T = 4$

1
3
5
10
13

10
13
15

5
7
13

13

15

结果: 13

序列合并算法

HeapMerger

MergeOpt

[SK04]
[LLL08, BK02]

ScanCount

MergeSkip

DivideSkip

基于堆的算法

Min-heap

输出 13

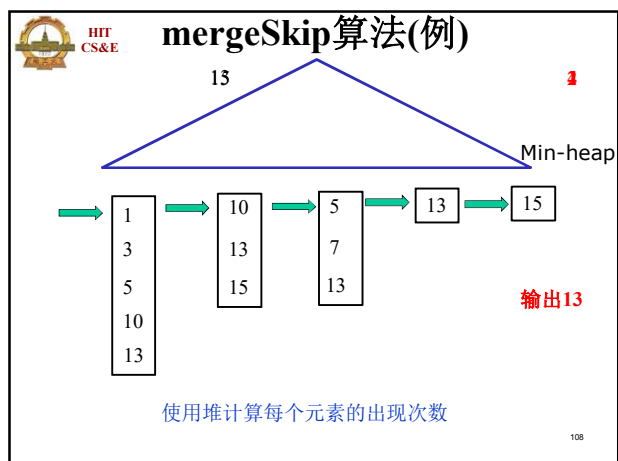
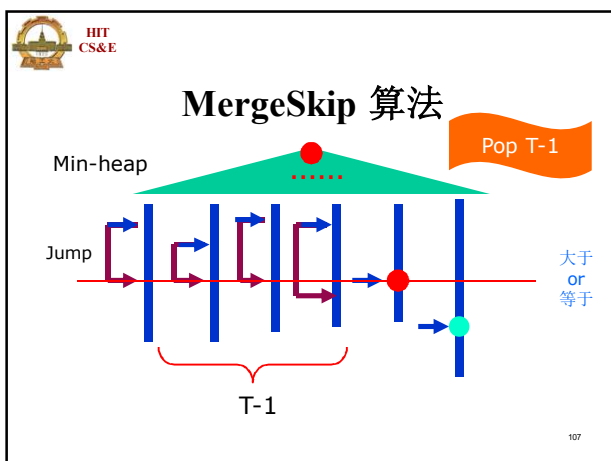
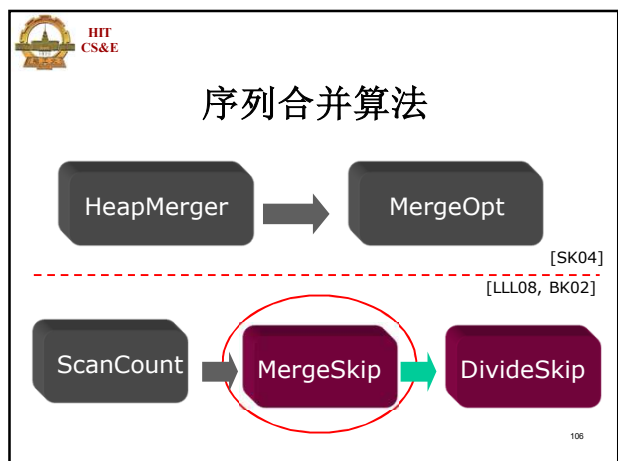
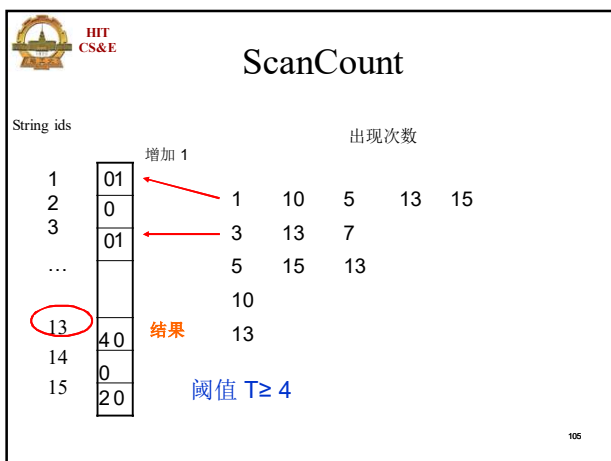
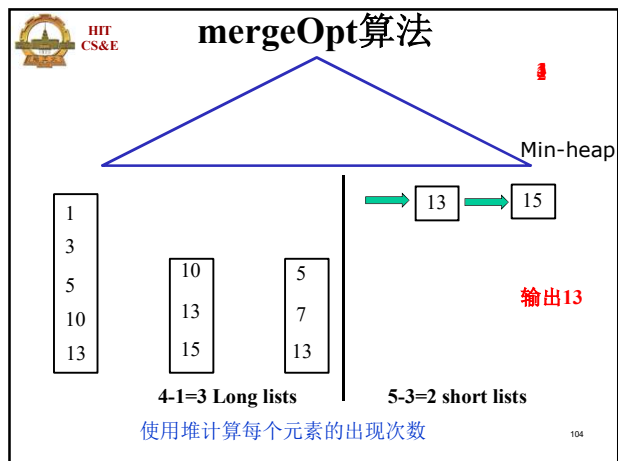
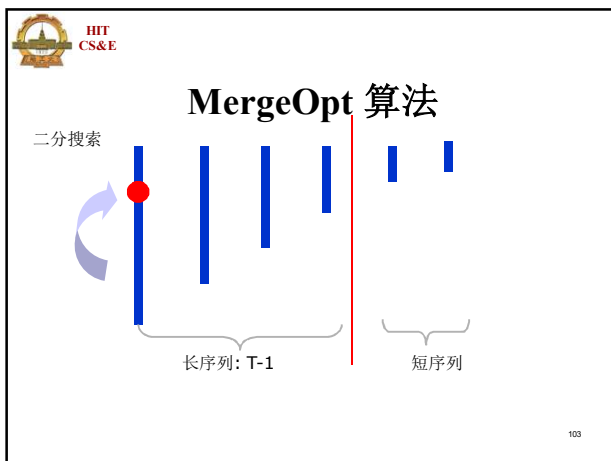
使用堆计算每个元素的出现次数

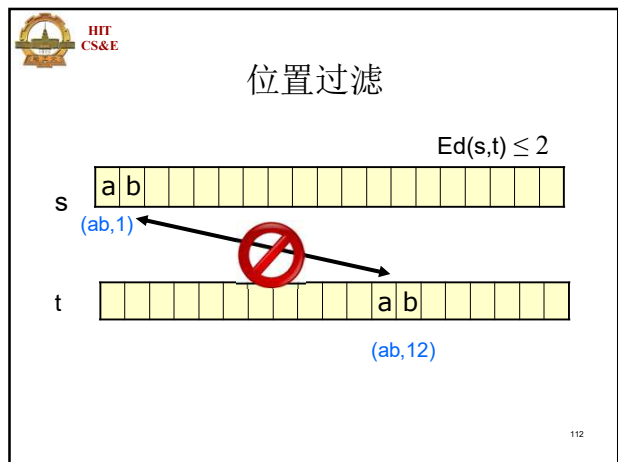
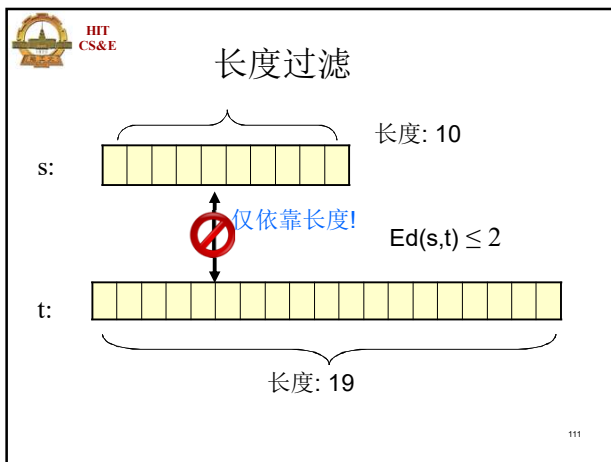
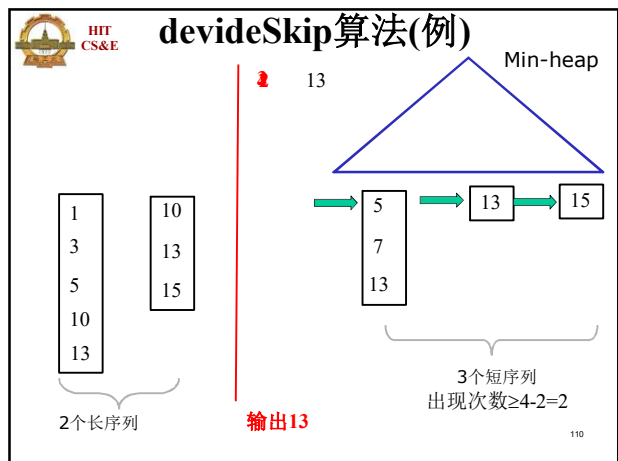
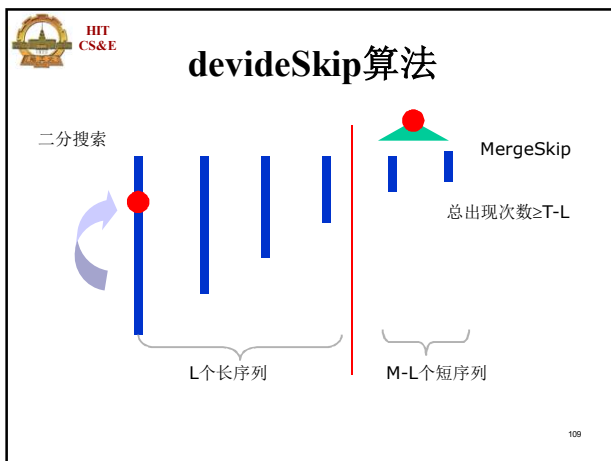
基于堆的算法

Push to heap

Min-heap

使用堆计算每个元素的出现次数





HIT CS&E

保持相似度的集合摘要

- Shingle 集合非常大。一个4 shingle 集合也是原始文件的4倍
- 想办法计算文件的签名（一个较小的文件），通过计算签名的相似性来推断文件之间的相似性。

HIT CS&E

集合的矩阵表示

- 矩阵的列表示各个集合，行表示所有可能的元素
- S

元素	s1	s2	s3	s4
A	1	0	0	1
B	0	0	1	0
C	0	1	0	1
D	1	0	1	1
e	0	0	1	0



最小哈希

- 首先选择行的一个排列变换。
- 任意一列的最小哈希值是在该行排列顺序下第一个列值为1的行的行号。
- $H(S1)=a, H(s2)=c, H(s3)=b, H(s4)=a$

元素	S1	S2	s3	s4
B	0	0	1	0
E	0	0	1	0
A	1	0	0	1
D	1	0	1	1
c	0	1	0	1



最小哈希及jaccard相似度

- 两个集合经随机排列转换之后得到的两个最小哈希值相等的概率等于这两个集合的jaccard相似度
- 假设只考虑s1和s2两个集合对应的列



四种类型

- ◆ 给定列 C_1 和 C_2 , 行可以如下划分:

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

- ◆ 两列同时为0的行对于这两列的相似性没有贡献。
- ◆ 设 x 是两列都为1的行的数目
- ◆ y 是其中一行为1的数目
- ◆ 注意 $Sim(C_1, C_2) = x / (x + y)$.



最小哈希

- 对行进行随机转换后, 从上往下扫描。遇到两行均为1的概率是 $x/(x+y)$
 - 总共有 $x+y$ 行, 两行均为1的个数有 x 个。
 - 两行均为1, 也就是 $H(s1)=H(s2)$



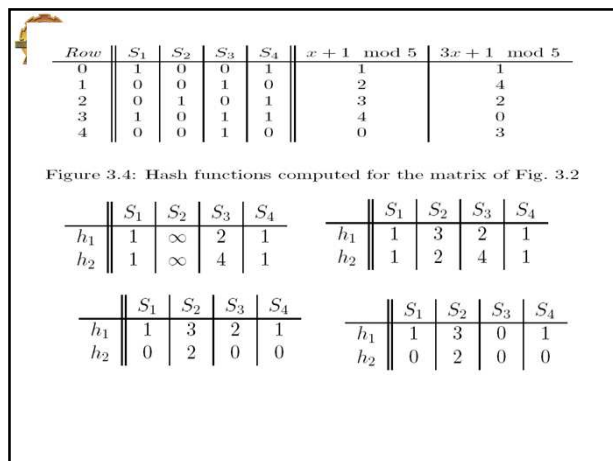
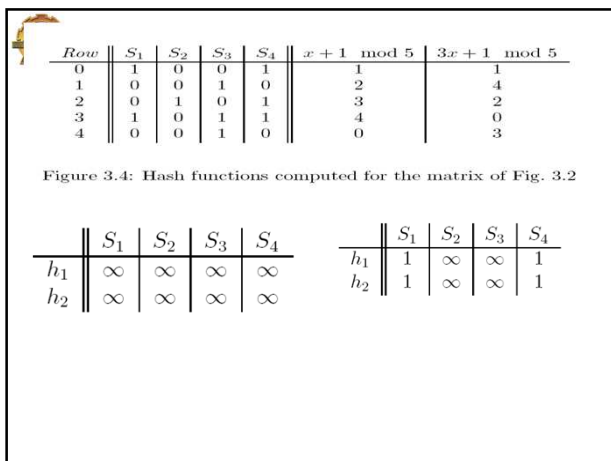
最小哈希签名

- 对于每一个行排列方式, 每一个集合都有一个最小哈希值
- 如果有 n 个行排列方式 (n 一般是1百到数百), 每个集合就能产生 n 个最小哈希值, 把这些哈希值写成一个列向量。也称为哈希签名。
- 每个集合的列向量组合在一起, 写成一个矩阵, 称为签名矩阵。
- 文档的相似性就等于最小哈希值相等的概率



最小哈希签名的计算

- 操作矩阵较为困难, 操作较小的签名矩阵是可以的
 - 用签名矩阵记录行变换的结果, 每一个位置只记录当前变换的最小的位置
1. Compute $h_1(r), h_2(r), \dots, h_n(r)$.
 2. For each column c do the following:
 - (a) If c has 0 in row r , do nothing.
 - (b) However, if c has 1 in row r , then for each $i = 1, 2, \dots, n$ set $SIG(i, c)$ to the smaller of the current value of $SIG(i, c)$ and $h_i(r)$.



HIT CS&E

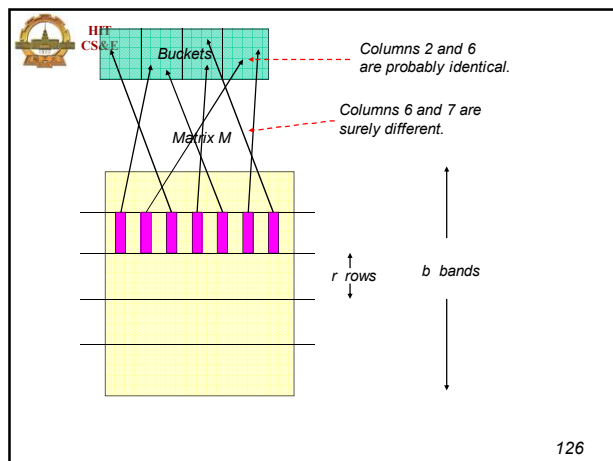
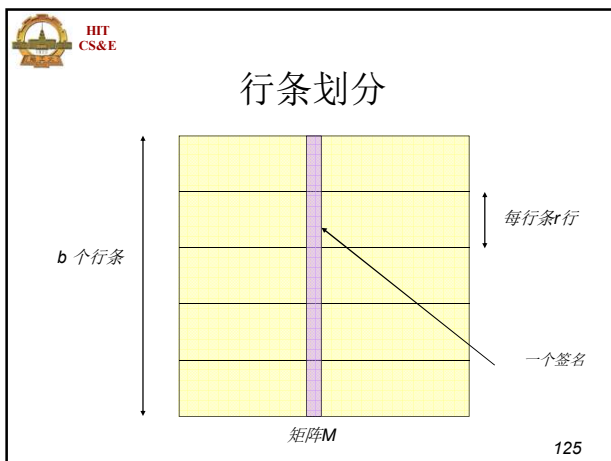
文档的局部敏感哈希算法

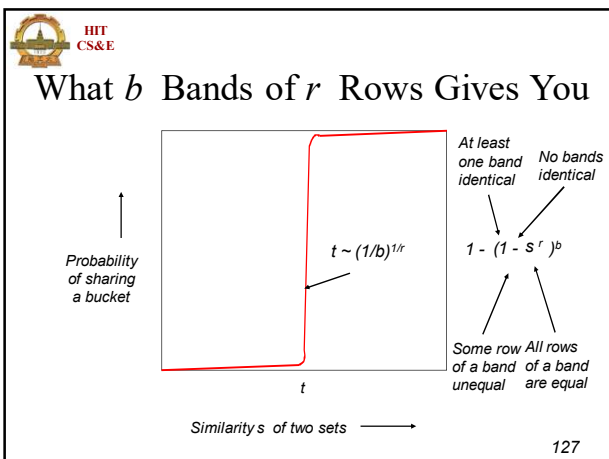
- 即使可以使用最小哈希将大文档压缩成小的签名并同时保存任意文档对之间的相似度，寻找相似文档对仍然是不可能的
 - 文档的数目太大
- 实际上，只需要寻找那些相似性大于某个阈值的文档对，而不是全部文档对。这就是局部敏感哈希

HIT CS&E

文档的局部敏感哈希算法

- 假设
 - 文档先表示为shingle集合，通过哈希处理，变为短签名集合
- 基本想法：
 - 把签名矩阵分成子矩阵，使用多次哈希函数。
 - 具有相同部分的列将被哈希到同一个桶中
 - 只考察那些哈希到同一个桶里面的列的相似性





HIT CS&E

Example: $b = 20; r = 5$

s	$1 - (1 - s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

128

-
- HIT CS&E
- ### LSH小结
- 找出具有相似签名的集合对，删除大部分不相似的集合对
 - 在内存里面检查这些候选的集合对
- 129

HIT CS&E

算法课没有结束.....

-
- HIT CS&E
- ### 当前的研究问题
- 近似算法
 - 随机算法
 - 图论算法
 - 计算几何
 - 经济/博弈算法
 - 算法复杂性
 - 分布式算法
 - NP难问题和P问题的近似算法
 - 实际应用中的问题
 - 经典问题的算法

-
- HIT CS&E
- ### 一些最新算法研究结果.....
- A Simpler and Faster Strongly Polynomial Algorithm for Generalized Flow Maximization
 - Simple Mechanisms for Subadditive Buyers via Duality
 - Faster Space-Efficient Algorithms for Subset Sum and k-Sum
 - Homomorphisms Are a Good Basis for Counting Small Subgraphs
 - Local Max-Cut in Smoothed Polynomial Time
 - Almost-Linear-Time Algorithms for Markov Chains and New Spectral Primitives for Directed Graphs
 - Uniform Sampling through the Lovasz Local Lemma
 - Sampling Random Spanning Trees Faster than Matrix Multiplication
 - Online and Dynamic Algorithms for Set Cover
 - Exponential Separations in the Energy Complexity of Leader Election
 - Decremental Single-Source Reachability in Planar Digraphs
 - A Strongly Polynomial Algorithm for Bimodular Integer Linear Programming