HIT
CS&E

第九章
字符串匹配算法

高宏
海量数据计算研究中心

HIT
CS&E

参考材料

《Introduction to Algorithm》

Chapter 32

HIT
CS&E

提要

HIT
CS&E

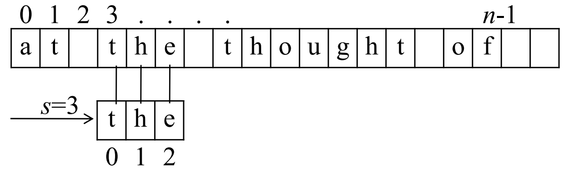
9.1 相关概念与问题定义

符号与术语

- Σ：字母表
- Σ^*：所有**有限长度**的字符串的集合，该字符串是由字母表Σ中的字母组成
  - ε表示长度为0的空字符串
  - $|x|$表示字符串x的长度
- 字符串的连接(concatenation)
  - 用xy表示字符串x和y的连接
    - 字符串x后接字符串y
  - $|xy|= |x|+ |y|$
- 前缀：若对于某个字符串y,有$x=wy$,则称w是x的前缀
- 后缀：若对于某个字符串y,有$x=yw$,则称w是x的后缀

问题定义

- 字符串匹配问题
  - 输入：字符串T和P，$|T|=n$，$|P|=m$，$(m\le n)$
    - 文本T = “at the thought of”
      - n = length(T) = 17
    - 模式 P = “the”
      - m = length(P) = 3
  - 输出：P在T中出现的位置
    - 所有s(偏移)
      - $0 \le s \le n-m$，满足 $T[s\ ..\ s+m-1] = P[0\ ..\ m-1]$
    - -1，如果不存在这样的s



HIT
CS&E

问题定义

- 单维模式匹配
  - 在一个文本text(设长度为n)中查找某个特定的子串pattern(设长度为m)。如果在text中找到等于pattern的子串，则称匹配成功，函数返回pattern在text中出现的位置(或序号)，否则匹配失败
- 多维模式匹配
  - 在一个文本text(设长度为n)中查找某些特定的子串patterns(设长度为m)。如果在text中找到等于patterns中的某些子串，则称匹配成功，函数返回pattern在text中出现的位置(或序号)，否则匹配失败

HIT
CS&E

9.2 精确匹配算法

HIT CS&E

一个直观的Naive算法

- 蛮力搜索(Brute Force，BF)
  - 在T的每个位置i，检查P是否匹配T的一个子串

T: ABABABCCA
P: ABABC

T: ABABABCCA
P: ABABC

T: ABABABCCA
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

时间复杂性$O((n-m-1)m)$

1. $n\leftarrow$length(T)
2. $m\leftarrow$length(P)
3. For $k\leftarrow0$ to n-m do
4. IF $P[1,\ldots,m] = T[k,k+1,\ldots,k+m-1]$ THEN print k

HIT CS&E

改进算法

- 1970年，S.A.Cook理论上证明了串匹配问题可以在O(m+n) 线性时间内解决，随后D.E.Knuth和V.R.Pratt仿照S.A.Cook的证明构造了一个算法，与此同时，J.H.Morris也得到相类似的算法，这样就产生了第一种线性时间复杂度的模式匹配算法
- 1977年，R.S.Boyer和J.S.Moore两人设计了一种新的算法（简称BM算法），该算法可以实现跳跃查寻，大多数情况下只需扫描文本中的一部分字符。尽管它的时间复杂度并不是最好，但是实际效果却常常是最快的。因此，BM算法得到了很好的研究，衍生出许多变种，如Horspool-BM, Tuned-BM和QS等，这些算法至今都是最活跃的算法。
- 1987年，Rabin和Karp提出算法，理想情况下期望时间复杂性为O(n-m)，并且算法可以推广至二维模式匹配。
- ……

Rabin-Karp算法

- 基本思想
  - 将字符串的比较转化成数的比较
  - 运用了初等数论：两个数相对于第三个数模等价
    - 若 $a \bmod q \equiv b \bmod q$，则$a$和$b$有可能相等，否则$a \neq b$
- $d=|\Sigma|$
  - Σ^*内任意字符串x可以看成一个d进制数
  - $P[0,2,\ldots,m-1]$可以看成

    $p = P[m-1]+ d(P[m-2]+ d(P[m-3]+\ldots+dP[0])\ldots)$
  - 类似地，$T[k,k+1,\ldots,k+m-1]$可以看成

    $t_k = T[k+m-1]+ d(T[k+m-2]+ d(T[k+m-3]+\ldots+dT[k])\ldots)$
- 如果$p \bmod q = t_k \bmod q$,则P在T中位置k处可能有一个匹配

Rabin-Karp算法

- 字符串数字化

P

0	1	2
t	h	e
20	8	5

$d=|\Sigma|=27$

$q=19$

$p=20*27^2+8*27+5 \bmod 19$ → $p=0$

T

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	15	21	7	8	20	0	15	6		

若在每个位置直接计算每个t_i，代价仍然是$(n-m+1)m$

$t_3= 20*27^2+8*27+5 \bmod 19 = 0$

$t_2= 0*27^2+20*27+8 \bmod 19 = 16$

$t_1= 20*27^2+0*27+20 \bmod 19 = 8$

$t_0= 1*27^2+20*27+0 \bmod 19 =15$

如何减少计算代价？
提高计算效率？

Rabin-Karp算法

- t_{k+1}与t_k之间的关系

$d=|\Sigma|=27$
$q=19$

P

0	1	2
t	h	e
20	8	5

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	15	21	7	8	20	0	15	6		

$T[2]$ $T[5]$

$t_3= 20*27^2+8*27+5 \bmod 19 = 0 = 27*t_2-27^3*0+5 \bmod 19$

$t_2= 0*27^2+20*27+8 \bmod 19 = 16 = 27*t_1-27^3*20+8 \bmod 19$

$t_1= 20*27^2+0*27+20 \bmod 19 = 8 = 27*t_0-27^3*1+20 \bmod 19$

$t_0= 1*27^2+20*27+0 \bmod 19 =15$

$T[4]$

$T[0]$ $T[3]$ $T[1]$

Rabin-Karp算法

$t_k = T[k+m-1]+ d(T[k+m-2]+ d(T[k+m-3]+\ldots+dT[k])\ldots)$

$\quad = d^{m-1}T[k]+ d^{m-2}T[k+1]+\ldots\ldots+T[k+m-1]$

$t_{k+1} = T[k+m]+ d(T[k+m-1]+ d(T[k+m-2]+\ldots+dT[k+1])\ldots)$

$\quad = d^{m-1}T[k+1]+ d^{m-2}T[k+2]+\ldots\ldots+ dT[k+m-1] +T[k+m]$

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

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

– 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

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. $h \leftarrow d^m$
4. $p \leftarrow 0$
5. $t_0 \leftarrow 0$
6. For $i \leftarrow 0$ to m-1 do //计算p和t_0
7. $p \leftarrow dp+P[i]$, $t_0 \leftarrow dt_0+T[i]$;
8. $p=p$ mod q;
9. for $k \leftarrow 0$ to n-m do
10. If $p= t_k$ mod q Then
11. If $P[1,\ldots,m] = T[k,k+1,\ldots,k+m-1]$ Then print k
12. $t_{k+1} \leftarrow dt_k - T[k]h + T[k+m]$

时间复杂度 $O((n-m-1)+cm)$

最坏时间复杂度 $O((n-m-1)m)$

$O(m)$

n-m-1遍

次数c

$O(m)$

$O(1)$

Rabin-Karp算法分析

若 q 是素数，求模计算会使m位字符串在q个值中均匀分布

因此，n-m次循环中仅在每第q次才需要匹配(匹配需要比较$O(m)$次)

理想的期望运行时间(如果 $q > m$)：

- 预处理：$O(m)$
- 外循环：$O(n$-$m)$
- 所有内循环：$\frac{n-m}{q} m = O(n-m)$
- 总时间：$O(n$-$m)$

最坏情况下：$O((n-m+1)m)$

9. for $k \leftarrow 0$ to n-m do
10. If $p=t_k$ Then
11. If $P[1,\ldots,m] = T[k,k+1,\ldots,k+m-1]$ Then print k
12.

Rabin-Karp比较简单，可以容易地拓展到2维模式匹配.

HIT
CS&E

有限自动机与字符串匹配

- 有限自动机就是构建出一个满足某个特定模式的判断系统
- 有限自动机M：$M(Q, q_0, A, \Sigma, \delta)$
  - Q：状态的有限集合；
  - $q_0 \in Q$：是初始状态； $A \in Q$：是接受状态集合
  - Σ是有限输入字母表
  - δ是状态转移函数：$Q\times \Sigma \rightarrow Q$
- 对于字符串的处理，我们可以利用有限自动机来判断。对模式串P构建一个有限自动机，用其来判断文本串T，如果文本串T可以到达YES的位置，说明文本串T中包含了模式串P

HIT
CS&E

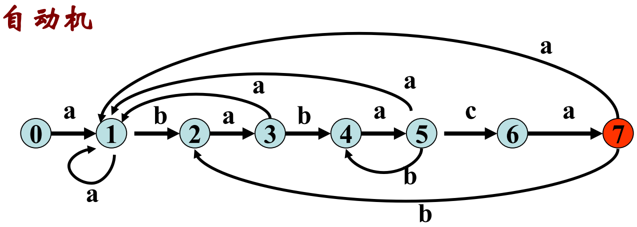
有限自动机与字符串匹配

T　a b a b a b a c a b a

P　a b a b a c a

对P构造有限自动机

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



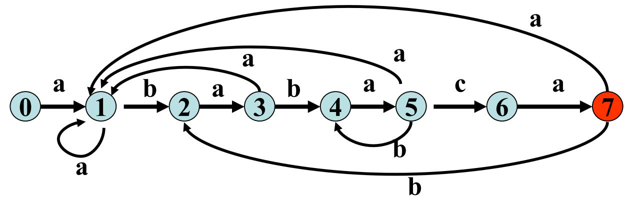
i		0	1	2	3	4	5	6	7	8	9	10
$T[i]$		a	b								b	
$\phi(T[i])$	0	1	2	3	4	5	4	5	6	7	2	

HIT CS&E

有限自动机与字符串匹配

如果FA存在,则找出所有匹配仅需对$T[1,2,\ldots,n]$线性扫描
关键是如何构造这个FA?

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



Finite-Autoation-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. 构造模式P的字符串匹配自动机FA
2. 逐个扫描文本T中的字符，记录当前状态
3. 如果 当前状态$q\in A$，找到一个 匹配

HIT CS&E

有限自动机与字符串匹配

对P=“ababaca”构造有限自动机

假定组成P和T的字符集Σ={a，b，c}，P含有m个字母，
于是我们要构建的自动机含有m个状态节点

设S_i后缀为当前读入的字符串，
$\sigma(S_i)$= P的前缀与所读入S_i后缀的最大匹配长度

定义状态转移函数：$\delta(q, i)=q' = \sigma(S_i)$

q=0 $\delta(0, \text{a})=\sigma(\text{“a”}) =1$
$\delta(0, \text{b})=\sigma(\text{“b”}) =0$
$\delta(0, \text{c})=\sigma(\text{“c”}) =0$

q=1 $\delta(1, \text{a})=\sigma(\text{“aa”}) =1$
$\delta(1, \text{b})=\sigma(\text{“ab”}) =2$
$\delta(1, \text{c})=\sigma(\text{“ac”}) =0$

	a	b	c	
0	1	0	0	a
1	1	2	0	b

HIT
CS&E

有限自动机与字符串匹配

对P="ababaca"构造有限自动机

假定组成P和T的字符集Σ={a，b，c}，P含有m个字母，
于是我们要构建的自动机含有m个状态节点

设S_i后缀为当前读入的字符串，
$\sigma(S_i)$= P的前缀与所读入S_i后缀的最长匹配长度

定义状态转移函数：$\delta(q, i)=q' = \sigma(S_i)$

	a	b	c	
0	1	0	0	a
1	1	2	0	b

q=2　$\delta(2, \text{a})=\sigma(\text{"aba"}) =3$
$\delta(2, \text{b})=\sigma(\text{"b"}) =0$
$\delta(2, \text{c})=\sigma(\text{"c"}) =0$

q=3　$\delta(3, \text{a})=\sigma(\text{"abaa"}) =1$
$\delta(3, \text{b})=\sigma(\text{"abab"}) =4$
$\delta(3, \text{c})=\sigma(\text{"abac"}) =0$

HIT
CS&E

有限自动机与字符串匹配

对P="ababaca"构造有限自动机

假定组成P和T的字符集Σ={a，b，c}，P含有m个字母，
于是我们要构建的自动机含有m个状态节点

设S_i后缀为当前读入的字符串，
$\sigma(S_i)$= P的前缀与所读入S_i后缀的最长匹配长度

定义状态转移函数：$\delta(q, i)=q' = \sigma(S_i)$

	a	b	c	
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b

q=2　$\delta(2, \text{a})=\sigma(\text{"aba"}) =3$
$\delta(2, \text{b})=\sigma(\text{"b"}) =0$
$\delta(2, \text{c})=\sigma(\text{"c"}) =0$

q=3　$\delta(3, \text{a})=\sigma(\text{"abaa"}) =1$
$\delta(3, \text{b})=\sigma(\text{"abab"}) =4$
$\delta(3, \text{c})=\sigma(\text{"abac"}) =0$

HIT CS&E

有限自动机与字符串匹配

- 算法复杂性分析

  算法分两个阶段
  - 预处理阶段：构建模式P对应的有限自动机；
    - $O(m|S|)$，需要的额外存储空间$O(m|S|)$
  - 匹配阶段： 扫描文本T
    - $O(n)$
- 问题：需要较大的存储空间

HIT CS&E

Knuth-Morris-Pratt 算法

- 第一种线性时间复杂性的匹配算法
- 算法基本思想

  充分利用已经比较过的字符信息来提高效率
- 已匹配成功部分的信息是可利用的
  - “前缀模式”
    - 模式中不同部分存在的相同子串
  - 根据前缀模式可以使模式向前推进若干字符位置
    - 依前缀模式长度而定
- 避免了重复比较，同时实现了文本的无回朔扫描

HIT
CS&E

Knuth-Morris-Pratt 算法

考察最简单最直接的扫描过程

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b	T
				a	b	a	b	a	c	a	P				

$\longrightarrow s \longrightarrow$ $\underbrace{ababa}_{q=5}$

–P_5已经被匹配上但P_6不匹配

–由于a≠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?

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b	T
						a	b	a	b	a	c	a	P		

$s'=s+2$

HIT
CS&E

Knuth-Morris-Pratt 算法

$\overbrace{ababa}^{q}$

b	a	c	b	a	b	a	b	a	a	b	c	b	a	b	T
s				a	b	a	b	a	P_q						
s'						a	b	a	P_k						

仅需将P_q与其自身进行比较

等价于找最大的k使得P_k是P_q的后缀

如果已知k的值，就可以按照下面步骤进行匹配

–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

问题的关键是：如何计算k，或P_k?

Knuth-Morris-Pratt 算法

- 模式P的前缀函数

  π: $\{1,2,\ldots,m\}\rightarrow\{1,2,\ldots,m\text{-}1\}$，满足

  $\pi(q)=\max\{k{:}\ k<q$, 且P_k是P_q的后缀$\}$

  – 即：$\pi(q)$ 是P_q的真后缀P的最长前缀长度

- 例如，给定模式P=“ababaca”

P	a	b	a	b	a	c	a
q	0	1	2	3	4	5	6
$\pi[q]$	0	0	1	2	3	0	1

可以预先计算大小为m的前缀表来存储$\pi[q]$的值 $(0 \le q < m)$

KMP-Matcher(P,T)

Input: 模式P,T

Output: P在T匹配的所有起始位置

```
m←length(P); n←length(T)
π←Compute_Prefex_Function(P);
q←0；                          //用于跟踪已匹配的字符数
For  i←2 to n do                //从左到右扫描T
    While  k>0  & P[q+1]≠T[i]  do
        q← π[q];
    If P[q+1]=T[i]  then   q←q+1;
    If  q=m   then
        print i-m+1;
        q ← π[q];
```

令状态q的势能为q,

经过平摊分析知：

计算前缀函数：$O(m)$

For循环匹配代价：$O(n)$

总的时间开销为$O(m+n)$

Boyer-Moore 算法

- 三种技巧
  - 逆向搜索：从P的后面开始搜索
  - 坏字符启发式规则
  - 好后缀启发式规则

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
```


Boyer-Moore 算法

```
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
```

$T[6] \neq P[6]$ 且 $T[6] \notin P$

P至少向后移动$|P|$个位置才可能发生匹配

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

HIT
CS&E

Boyer-Moore 算法

- 坏字符规则

  若$P[j] \neq T[i]$时，

  – 如果坏字符$T[i]$没有出现在模式P中，则直接将模式串移动到坏字符的下一个字符

  – 如果坏字符$T[i]$出现在模式P中

    – $a=\max\{q: P[q]=T[i]$，$P[q]$位于$P[j]$左侧$\}$，将模式串P右移$j-a$位

  即：如果$T(i)=x$，将P中位置j左面最近的x移到$T(i)$下面

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
H E R E   I S   A   S I M P L E   E X A M P L E
                        E X A M P L E
                        0 1 2 3 4 5 6
```

$T[13] \neq P[6]$，但$T[13]=P[4]$，P至少向后移动6-4=2个位置，才可能发生匹配

HIT
CS&E

Boyer-Moore 算法

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
H E R E   I S   A   S I M P L E   E X A M P L E

                          E X A M P L E
                          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
H E R E   I S   A   S I M P L E   E X A M P L E
                        E X A M P L E
                        0 1 2 3 4 5 6
```

HIT
CS&E

Boyer-Moore 算法

- 坏字符规则
  - 需要额外存储空间，记录模式P中每个字符出现的位置信息
    - 如：一个一维数组
    - P=abacbabc 那么R(a)={6,3,1}，所需空间$O(n)$
  - 局限性
    - 不适合小字符表
    - 最坏情况下时间复杂性非线性

如何利用已经匹配的子串？

HIT
CS&E

Boyer-Moore 算法

- 好后缀
  - 从后向前扫描遇到“坏字符”前已经匹配的子串

```
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[6-6]
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[3-6]
出现位置：6

Boyer-Moore 算法

- 好后缀
  - 从后向前扫描遇到“坏字符”前已经匹配的子串

例 1 2 3 4 5 6 7 8 9 0 1 2 3 4
T p r s t a b s t u b a b v q x r
P q c a b d a b d a b
1 2 3 4 5 6 7 8 9 0

因为*P*(8)不等于*T*(10)，依据坏字符规则使*P*向右移动1步

1 2 3 4 5 6 7 8 9 0 1 2 3 4
T p r s t a b s t u b a b v q x r
P q c a b d a b d a b
1 2 3 4 5 6 7 8 9 0

问题？

如何移动才是最佳的？

Boyer-Moore 算法

- 好后缀规则
  - 给定*P*、*T*，*T*的子串*t*匹配了*P*的一个后缀，但是再往左一个字符就不匹配了。寻找*t'*：

1. *t'*和*t*相同
2. *t'*不是*P*的后缀
3. *t'*左面的那个字符同与*t*匹配的*P*的后缀的左面那个字符不相同

例 1 2 3 4 5 6 7 8 9 0 1 2 3 4
T p r s t a b s t u b a b v q x r
P q c a b d a b d a b
1 2 3 4 5 6 7 8 9 0

HIT
CS&E

Boyer-Moore 算法

- 好后缀规则
  - 给定P、T，T中字符t后面子串s匹配了P的一个后缀，但是再往左一个字符就不匹配了。寻找t'：
    - t'和t相同
    - t'不是P的后缀
    - t'左面的那个字符同与t匹配的P的后缀的左面那个字符不相同
  - 将P向右移，直到t'位于t的下面

```
例 1 2 3 4 5 6 7 8 9 0 1 2 3 4
T  p r s t a b s t u b a b v q x r
P        q c a b d a b d a b
         1 2 3 4 5 6 7 8 9 0
```

```
例 1 2 3 4 5 6 7 8 9 0 1 2 3 4
T  p r s t a b s t u b a b v q x r
P                    q c a b d a b d a b
                       1 2 3 4 5 6 7 8 9 0
```

好后缀=ab，
若P向右移动6步可以得到最好的匹配效率

HIT
CS&E

Boyer-Moore 算法

- 好后缀规则
  - 给定P、T，T中字符t后面子串s匹配了P的一个后缀，但是再往左一个字符就不匹配了。寻找t'：
    - t'和t相同
    - t'不是P的后缀
    - t'左面的那个字符同与t匹配的P的后缀的左面那个字符不相同
  - 将P向右移，直到t'位于t的下面
  - 若不存在t'，则将P向右移动最少的步数，使P的前缀同t右子串s的后缀相匹配（与KMP类似）

```
例 1 2 3 4 5 6 7 8 9 0 1 2 3 4
T  p r s t a b s t u b a b v q x r
P            a b d a b d a b
             1 2 3 4 5 6 7 8
```

➡

```
例 1 2 3 4 5 6 7 8 9 0 1 2 3 4
T  p r s t a b s t u b a b v q x r
P                a b d a b d a b
                 1 2 3 4 5 6 7 8
```

HIT
CS&E

Boyer-Moore 算法

- 好后缀规则
  - 给定P、T，T中字符t后面子串s匹配了P的一个后缀，但是再往左一个字符就不匹配了。寻找t'：
    - t'和t相同
    - t'不是P的后缀
    - t'左面的那个字符同与t匹配的P的后缀的左面那个字符不相同
  - 将P向右移，直到t'位于t的下面
  - 若不存在t'，则将P向右移动最少的步数，使P的前缀同t右子串s的后缀相匹配（与KMP类似）
  - 若这种情况也不存在，则将P右移m步（与KMP类似）
  - 如果T中发现了一个P，将P向右移动最少格数使：P的前缀能够和T中发现的P的后缀相匹配。如果没有这种匹配，则将P向后移动m位（与KMP类似）

HIT
CS&E

Boyer-Moore 算法

- 好后缀规则
  - 问题：预处理太复杂
  - Horspool在1980年改进并简化了BM算法

HIT
CS&E

Boyer-Moore-Horspool 算法

- 在移动模式P时，仅考虑坏字符策略
  - 首先比较文本指针所指字符和模式P的最后一个字符，如果相等再比较其余的m-1个字符。
- 研究表明：坏字符的启发式规则在匹配过程中占主导地位的概率为94.03%，远高于好后缀的规则
- 因此，一般情况下，BMH算法比BM算法有更好的性能。它简化了初始化的过程。

HIT
CS&E

9.3 一些重要的数据结构

HIT
CS&E

- 预处理 P
  - 基于有限自动机的字符串匹配算法
  - Knuth-Morris-Pratt
  - Boyer-Moore
  - ……
- 预处理T
  - 如何组织文本字符串，提高匹配效率
  - 以空间换时间

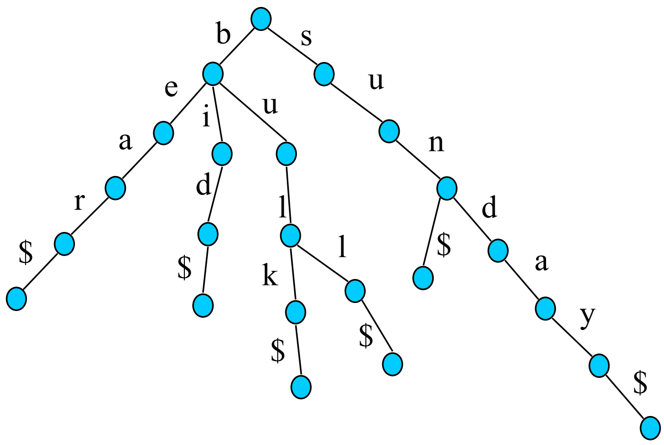
HIT
CS&E

Trie 树

- 字典树(前缀树、单词查找树、)
  - 一种多叉树形结构
  - Trie这个术语来自于re*trie*val.
- Trie树三个基本性质
  - 每个结点有从1到d个儿子
  - 每条边有一个字符做标记
  - 每个节点存储的字符串是从根到该节点路径上所有字符的连接

HIT
CS&E

Trie 树

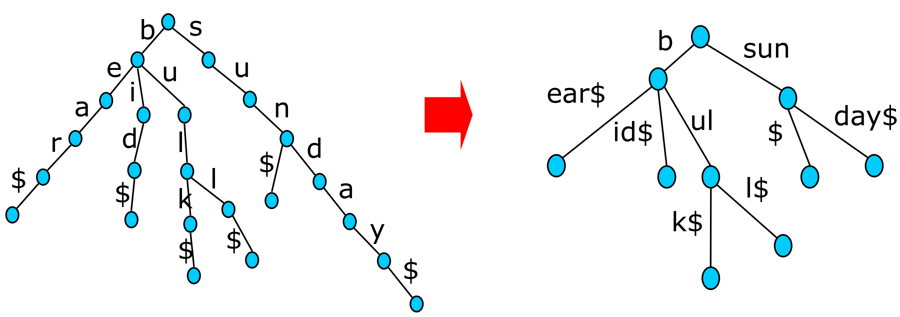


字符串集合: {**bear, bid, bulk, bull, sun, sunday**}
假设每个字符串以“$”(不在S中)结束

HIT
CS&E

紧缩Trie 树

- 用带有字符串的边取代一系列单儿子结点构成的链
- 每个非叶结点最少有两个儿子



HIT
CS&E

后缀

S = b b a b

S[1…8]= b b a b b a a b　　1-后缀

S[2…8]= b a b b a a b　　2-后缀

S[3…8]= a b　　3-后缀

S[4…8]= b b a a b　　4-后缀

S[5…8]= b a a b　　5-后缀

S[6…8]= a a b　　6-后缀

S[7…8]= a b　　7-后缀

S[8…8]= b　　8-后缀

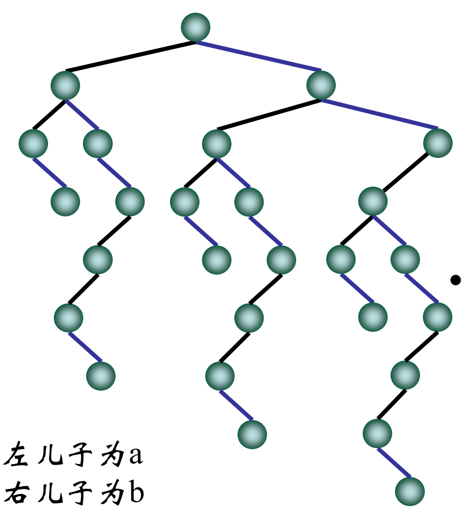
注意：*P*在*S*中出现当且仅当*P*是某个*i*-后缀的前缀

所谓预处理就是恰当地组织后缀

HIT
CS&E

后缀Trie树

T = S的后缀树



左儿子为a
右儿子为b

S的所有后缀

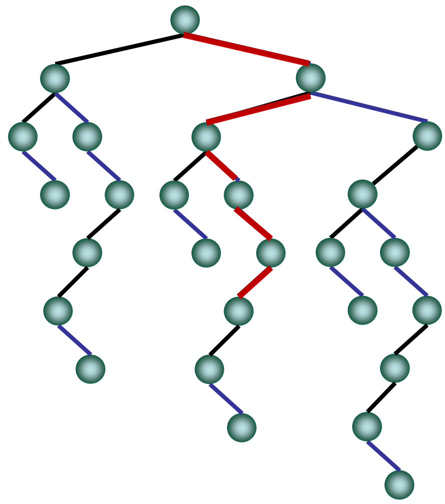
b b a b b a a b
b a b b a a b
a b b a a b
b b a a b
b a a b
a a b
a b
b

- 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

后缀Trie树

T = S的后缀树



P = b a b b a

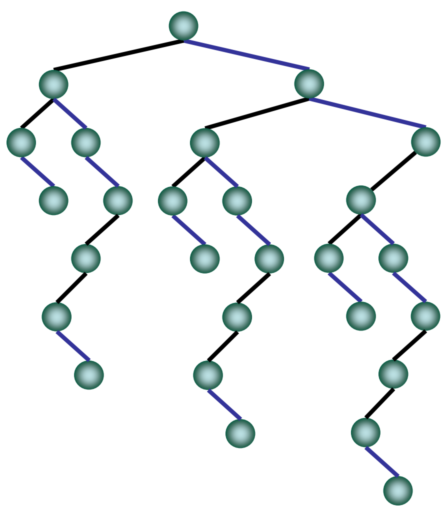
b b a b b a a b
b a b b a a b
a b b a a b
b b a a b
b a a b
a a b
a b
b

P在S中出现

HIT
CS&E

后缀Trie树

T = S的后缀树



P = b b a a b a

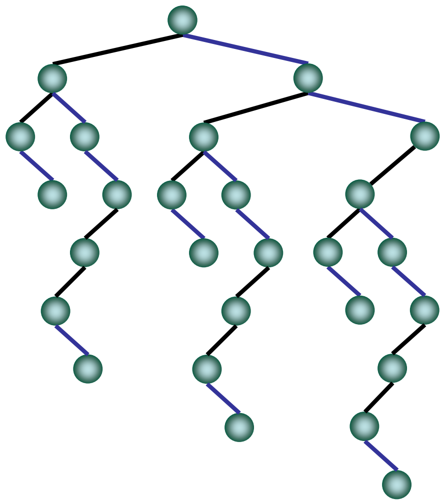
b b a b b a a b
b a b b a a b
a b b a a b
b b a a b
b a a b
a a b
a b
b

P未在S中出现

HIT
CS&E

后缀Trie树

T = S的后缀树



P = a b b b a a

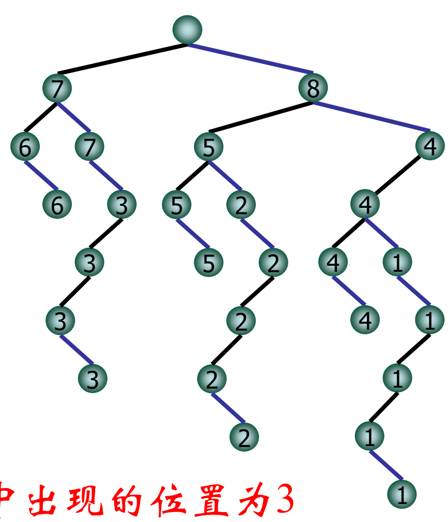
b b a b b a a b
b a b b a a b
a b b a a b
b b a a b
b a a b
a a b
a b
b

*P*未在*S*中出现

HIT
CS&E

后缀Trie树

P在S中出现的位置?



P = a b b a a

b b a b b a a b
b a b b a a b
a b b a a b
b b a a b
b a a b
a a b
a b
b

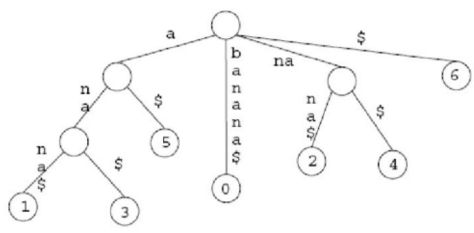
*P*在*S*中出现的位置为3

HIT
CS&E

后缀树

- 后缀紧凑Trie树
  - 一种包含文本所有后缀的紧缩trie(或类似的结构)
  - 只存储分叉节点和叶子
  - 边上的标记在逻辑上了变成字符串

banana对应的后缀树



HIT
CS&E

倒排索引(Inverted Index)

- 一个简单的例子(金庸小说)
  - 金庸的哪本小说包含郭靖和黄蓉但不包含洪七公?
    - 布尔表达式为 “郭靖 AND 黄蓉 AND NOT 洪七公”
- 直观做法
  - 从头到尾扫描所有小说，对每本小说判断它是否包含郭靖和黄蓉但不包含洪七公

HIT
CS&E

倒排索引(Inverted Index)

词项-文档(term-doc)的关联矩阵

	射雕英雄传	神雕侠侣	天龙八部	倚天屠龙记	鹿鼎记
郭靖	1	1	0	1	0
黄蓉	1	1	0	1	0
洪七公	1	1	0	0	0
张无忌	0	0	0	1	0
韦小宝	0	0	0	0	1

Query: 郭靖 *AND* 黄蓉 *BUT NOT* 洪七公?
倚天屠龙记!

若某小说包含某单词，则该位置置为1，否则为0

HIT
CS&E

倒排索引(Inverted Index)

- 假定N = 1 百万篇文档(1M), 每篇有1000个词
- 假定每个词平均有6个字节(包括空格和标点符号)
  - 那么所有文档将约占6GB 空间.
- 假定 词汇表的大小(即词项个数) |V| = 500K
  - 词项-文档矩阵的大小为 500K x 1M=500G
  - 然而，该矩阵中最多有10亿(1G)个1
    - 词项-文档矩阵高度稀疏(sparse).
- 是否有更好的表示方法?

Why?

HIT
CS&E

倒排索引(Inverted Index)

- 更好的表示
  - 仅仅记录所有1的位置
  - 对每个词项*t*, 记录所有包含*t*的文档列表
    - 每篇文档用一个唯一的 docID来表示，通常是正整数，如1,2,3…
  - 若采用定长数组的方式来存储docID列表，浪费存储空间

Brutus →	1	2	4	11	31	45	173	174
Caesar →	1	2	4	5	6	16	57	132
Calpurnia →	2	31	54	101				

HIT
CS&E

倒排索引(Inverted Index)

- 倒排索引

Doc 1: I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2: So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

词项及文档频率

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

docID表

addressing granularity: inverted file document

指针

HIT
CS&E

倒排索引(Inverted Index)

- 倒排索引

1 6 9 11 17 19 24 28 33 40 46 50 55 60

This is a text. A text has many words. Words are made from letters.

Text

Vocabulary	Occurrences
letters	60 …
made	50 …
many	28 …
text	11, 19 …
words	30, 40 …

addressing granularity:
(1) inverted list –
word positions
character positions
(2) inverted file document

HIT
CS&E

倒排索引(Inverted Index)

- 倒排索引
  - 块地址索引
  - 对每个词项t，记录所有包含t的文档列表
    - 有时候为了节省索引空间，可按块地址建索引
    - 把原文划分为多个块，只记录关键词的块地址

Block1	Block2	Block3	Block 4
This is a text.	A text has many	words. Words are	made from letters.

Text

Vocabulary	Occurrences
letters	4 …
made	4 …
many	2 …
text	1, 2 …
words	3 …

Inverted index

HIT
CS&E

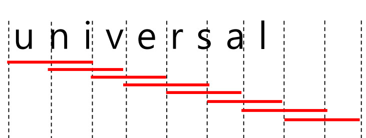
倒排索引(Inverted Index)

- 词汇表文件的组织方式
  - 采用Hash散列表
  - 按字母表顺序有序排列
  - 采用Trie树、B树等查找树
- 置入文件的压缩
  - 通常采用差值压缩（delta compression）

HIT
CS&E

q-Gram

"q-grams"

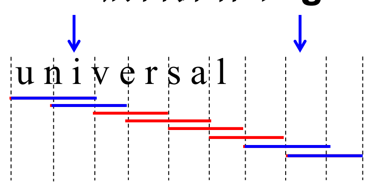


2-grams

HIT
CS&E

q-Gram

编辑操作和 gram的关系



固定长度: q

k 个操作会影响 $k * q$ 个gram

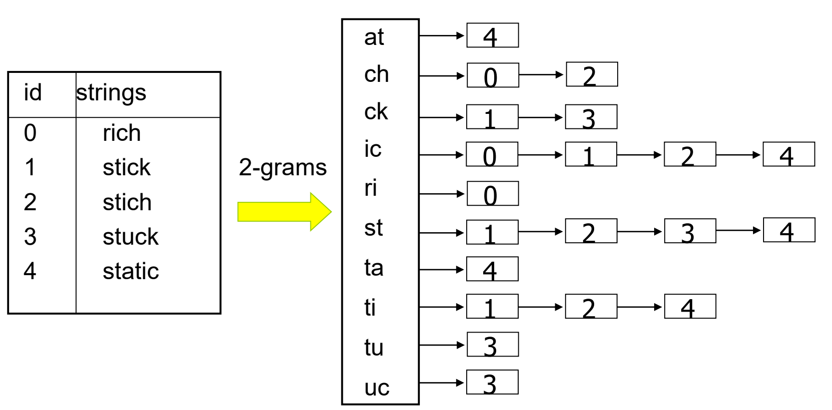
如果ed(s1,s2) <= k, 那么他们公共gram的数量 >=

$(|s_1| - q + 1) - k * q$

HIT
CS&E

q-Gram

q-gram 的倒排表



HIT
CS&E

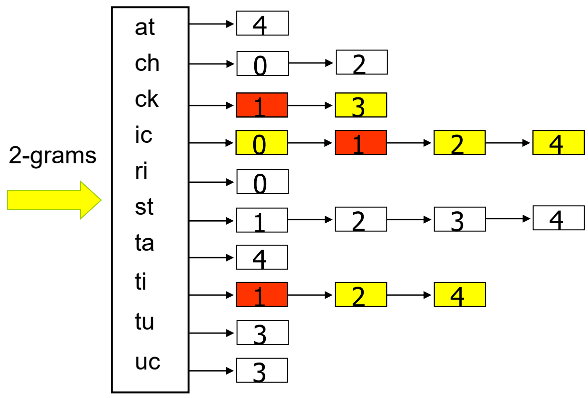
q-Gram

使用倒排表的搜索

- 查询: “shtick”, ED(shtick, ?)≤1　　公共 gram的数量 >= 3

  sh　ht　ti　ic　ck

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



65

6/5/2019

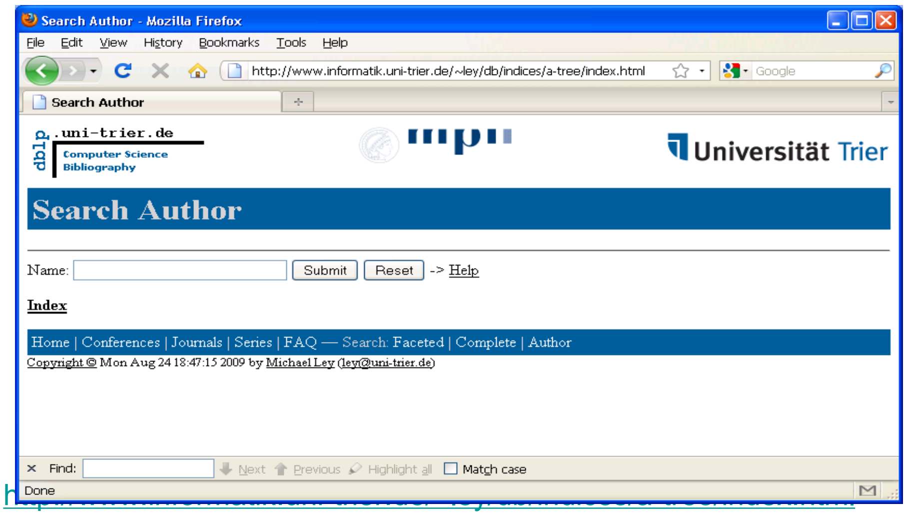
HIT
CS&E

9.4 近似匹配算法

HIT
CS&E

Why?

DBLP 作者搜索



67

6/5/2019

HIT
CS&E

Why?

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

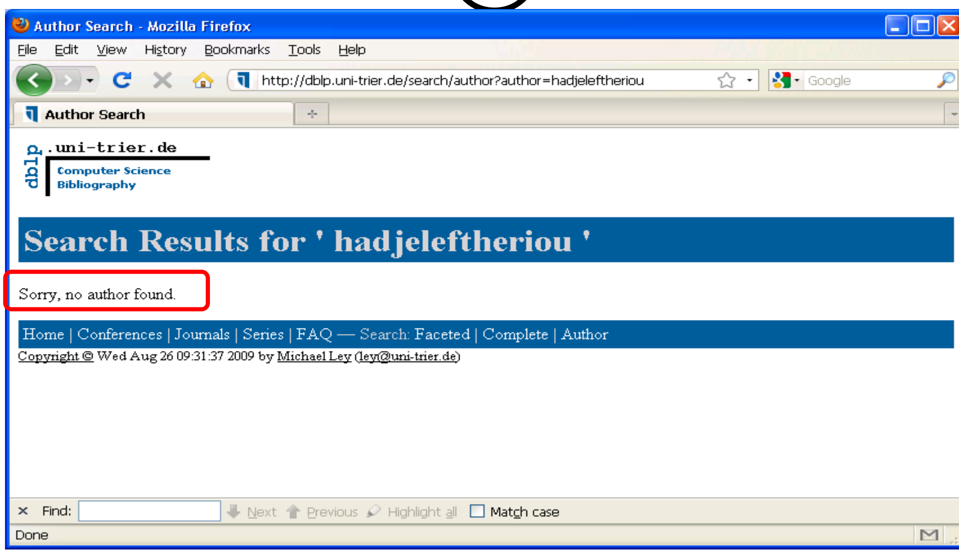
68

6/5/2019

HIT
CS&E

Why?

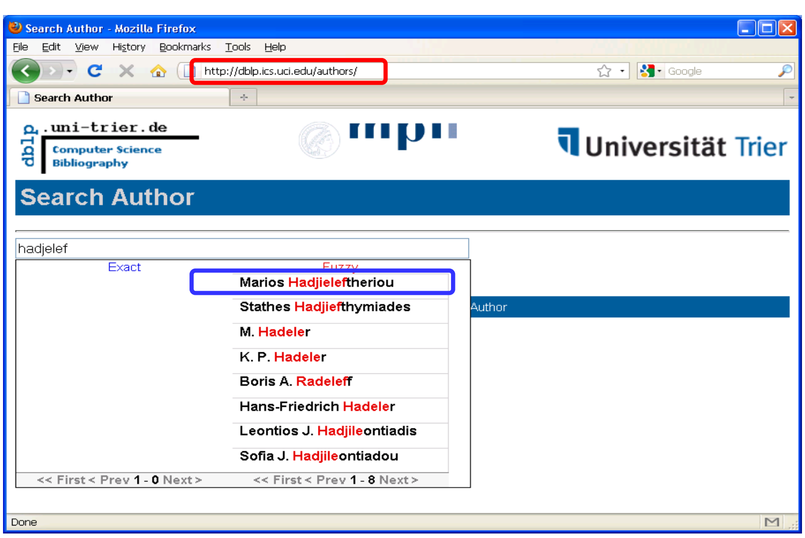
☹



HIT
CS&E

Why?

更好的系统?



http://dblp.ics.uci.edu/authors/

HIT
CS&E

Why?

UC Irvine的人名搜索

http://psearch.ics.uci.edu/

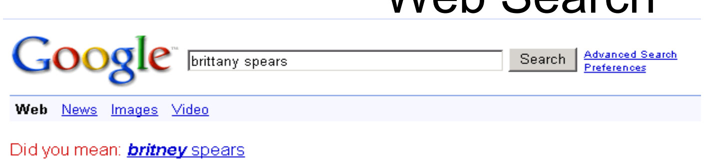
71

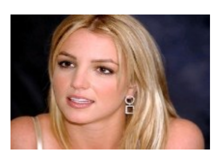
6/5/2019

HIT
CS&E

Why?

Web Search





```
488941 britney spears
 40134 brittany spears
 36315 brittney spears
 24342 britany spears
  7331 britny spears
  6633 briteny spears
  2696 britteny spears
  1807 briney spears
  1635 brittny spears
  1479 brintey spears
  1479 britanny spears
  1338 britiny spears
  1211 britnet spears
  1096 britiney spears
```

Google搜集的实际查询

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

- 查询有错误
- 数据有错误
- 搜索与查询相接近的结果

72

6/5/2019

HIT
CS&E

Why?

数据清洗

R

informix
microsoft
…
…

S

infromix
...
mcrosoft
…

HIT
CS&E

问题定义

查找和给定字符串相似的字符串: dist(Q,D) <= δ

例: 找到和"hadjeleftheriou"相似的字符串

性能很重要!

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

- 5 ms: 200 QPS

相似性函数

- 领域相关的函数
- 返回字符串间的相似性值
- 例如：
  - 编辑距离
  - Hamming距离
  - Jaccard距离
  - ……

相似性函数

- 编辑距离
  - 一种广泛使用的字符串相似性测度

    Ed(s1,s2) =将s1变化到s2需要的最小操作数

    (增加、删除、修改)

例如： s1: Tom Hanks

s2: Ton Hank

ed(s1,s2) = 2

HIT CS&E

相似性函数

- Hamming距离(海明距离)
  - 两个长度相等的字符串的海明距离是在相同位置上不同的字符的个数，也就是将一个字符串替换成另一个字符串需要的替换的次数

例如：

"toned" 与 "roses" 之间的汉明距离是3

HIT CS&E

相似性函数

- Jaccard距离
  - Jaccard系数：度量两个集合A和B之间的相似性

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

  - Jaccard距离：度量两个集合之间的差异性

$$d_J(A,B)=1-J(A,B)=\frac{|A\cup B|-|A\cap B|}{|A\cup B|}$$

HIT
CS&E

k-近似匹配

- 问题定义
  - 输入：给定长度为n的文本T，长度为m的模式P，以及非负整数k
  - 输出：P在T中的k-近似匹配，即P在T中的包含至多k个差别的匹配
    - 修改：P与T中对应的字符不同
    - 删除：T中含有一个未出现在P中的字符
    - 插入：T中不含有出现在P中的字符

```
P:a p p r o x i   m a t l y

T:a p   r o x i o m a l l y
```

HIT
CS&E

k-近似匹配

- 问题定义
  - 在精确串匹配中，P在T上的一次匹配检查十分简单，用至多 m 次字符比较即可完成，问题的关键是确定一次匹配检查之后模式P的移动方法。
  - k-近似匹配的关键是P在T上的k-近似匹配检查，即计算模式 P与当前对应的文本 T之子串的最小差别数。如果该值小于等于k,则找到结果，否则模式P右移。
    - 计算集中在计算最小差别数上，这实际上是一个优化问题，比精确匹配问题复杂得多。

k-近似匹配

- 近似串匹配的一个简化问题是比较两个字符串的差别。
  - 动态规划算法:从右向左搜索
    - 近似串匹配问题同样具有最优子结构性质和子问题重叠性质

HIT
CS&E

k-近似匹配

- 定义一个代价函数 $D[i][j], 0 \le i \le m, 0 \le j \le n$。

  $D[i][j]$表示模式子串$p_1 \ldots p_i$与文本子串$t_1 \ldots t_j$之间的最小差别。

  $D[m][j]$表示模式P 在文本T的位置j处的最小差别，如果$D[m][j] \le k$, 说明P在t_j处找到了k-近似匹配。

代价函数的初始值：

$D[0][j]=0$,这是因为模式 P 为空串，与文本 $t_1 \ldots t_j$ 有 0 个字符不同；

$D[i][0]=i$,这是因为模式串 $p_1 \ldots p_i$与空文本串相比，有i个字符不同。

k-近似匹配

对于模式子串$p_1...p_i$与文本子串$t_1...t_j$，有四种可能的情况：

(1) 若$p_i = t_j$， D[i][j]=D[i-1][j-1];

(2) 若$p_i \neq t_j$， D[i][j]= D[i-1][j-1]+1; /修改t_j为p_i

(3) 若删除t_j，D[i][j]= D[i][j-1]+1;

(4) 若在t_j后插入p_i，D[i][j]= D[i-1][j]+1;

代价函数的递归方程：

D[i][j]=D[i-1][j-1]   若$p_i = t_j$

D[i][j]= min{D[i-1][j-1]+1, D[i][j-1]+1, D[i-1][j]+1}，否则