CS481/CS583: Bioinformatics Algorithms

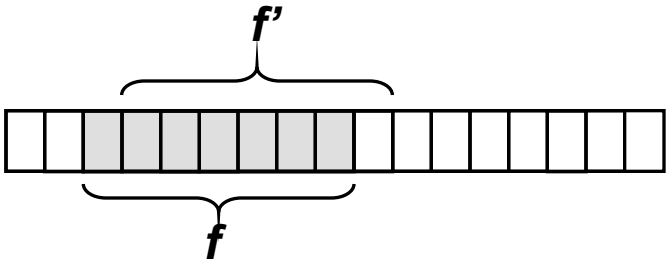
Can Alkan
EA509
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/

RABIN-KARP ALGORITHM

Fingerprint idea

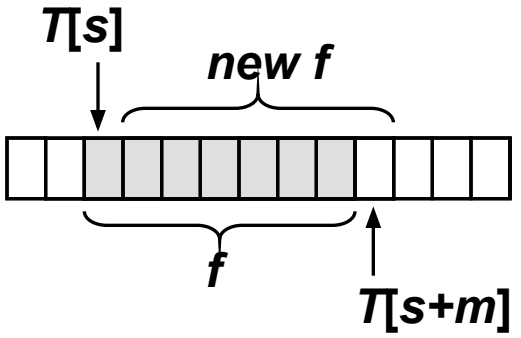
- Assume:
  - We can compute a fingerprint f(P) of P in O(m) time.
  - If f(P) ≠ f(T[s .. s+m–1]), then P ≠ T[s .. s+m–1]
  - We can compare fingerprints in O(1)
  - We can compute f' = f(T[s+1.. s+m]) from f(T[s .. s+m–1]), in O(1)



Algorithm with Fingerprints

- Let the alphabet $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$
- Let fingerprint to be just a decimal number, i.e., $f(\text{“1045”}) = 1*10^3 + 0*10^2 + 4*10^1 + 5 = 1045$

```
Fingerprint-Search(T,P)
01 fp ← compute f(P)
02 f ← compute f(T[0..m−1])
03 for s ← 0 to n − m do
04     if fp = f return s
05     f ← (f − T[s]*10^(m−1))*10 + T[s+m]
06 return −1
```



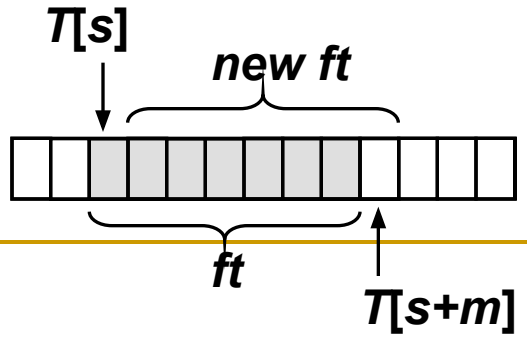
- **Running time $2O(m) + O(n–m) = O(n)$**

Using a Hash Function

- Problem:
  - we can not assume we can do arithmetics with m-digits-long numbers in O(1) time
- Solution: Use a hash function h = f mod q
  - For example, if q = 7, h("52") = 52 mod 7 = 3
  - h(S1) ≠ h(S2) ⇒ S1 ≠ S2
  - But h(S1) = h(S2) does not imply S1=S2
    - For example, if q = 7, h("73") = 3, but "73" ≠ "52"
- Basic "mod q" arithmetics:
  - (a+b) mod q = (a mod q + b mod q) mod q
  - (a*b) mod q = (a mod q)*(b mod q) mod q

Preprocessing and Stepping

- Preprocessing:
  - fp = P[m-1] + 10*(P[m-2] + 10*(P[m-3]+ … + 10*(P[1] + 10*P[0])…)) mod q
  - In the same way compute ft from T[0..m-1]
  - Example: P = “2531”, q = 7, fp = ?
- Stepping:
  - ft = (ft – T[s]*10^{m-1}mod q)*10 + T[s+m]) mod q
  - 10^{m-1} mod q can be computed once in the preprocessing
  - Example: Let T[…] = “5319”, q = 7, what is the corresponding ft?



Stepping

- T = 25316446766…, m = 4, q=7
- T_0 = "2531"
  - ft = 2531 mod 7 = 4
- T_1 = "5319"
  - ft = ((ft – T[s]*(10^{m-1} mod q))*10 + T[s+m]) mod q
  - ft = ((ft – T[0]*(10^3 mod 7))*10 + T[0+4]) mod 7
    - = ((4 – (2*1000 mod 7)) * 10 + T[4]) mod 7
    - = ((4-(2*6))*10+6) mod 7 = (-8*10+ 9) mod 7
    - = -71 mod 7 = 6
  - 5319 mod 7 = 6

Rabin-Karp Algorithm

```
Rabin-Karp-Search(T,P)
01 q ← a prime larger than m
02 c ← 10^(m-1) mod q  // run a loop multiplying by 10 mod q
03 fp ← 0; ft ← 0
04 for i ← 0 to m-1  // preprocessing
05    fp ← (10*fp + P[i]) mod q
06    ft ← (10*ft + T[i]) mod q
07 for s ← 0 to n − m  // matching
08    if fp = ft then   // run a loop to compare strings
09       if P[0..m-1] = T[s..s+m-1] return s
10    ft ← ((ft − T[s]*c)*10 + T[s+m]) mod q
11 return −1
```

Analysis

- If q is a prime, the hash function distributes m-digit strings evenly among the q values
  - Thus, only every q^{th} value of shift s will result in matching fingerprints (which will require comparing strings with O(m) comparisons)
- Expected running time (if q > m):
  - Preprocessing: O(m)
  - Outer loop: O(n-m)
  - All inner loops: $\frac{n-m}{q} m = O(n-m)$
  - Total time: O(n-m)
- Worst-case running time: O(nm)

Rabin-Karp in Practice

- If the alphabet has d characters, interpret characters as radix-d digits (replace 10 with d in the algorithm).
- Choosing prime q > m can be done with randomized algorithms in O(m), or q can be fixed to be the largest prime so that 10*q fits in a computer word.

RABIN-KARP – EXAMPLE #2

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$$c = 4^{4-1} mod\ 997 = 4^3 = 64$$

Fingerprint(P) = code(G) + 4*code(A) + 16*code(T) + 64*code(C)
= 2 + 4*0 + 16*3 + 64*1 = 114

Fingerprint(T) = code(G) + 4*code(C) + 16*code(A) + 64*code(T)
= 2 + 4*1 + 16*0 + 64*3
= 198

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 198

i = 2
Fingerprint(T) = (F(T) – T[1]*c) * 4 + T[5]
= (198 – code(T)*64) * 4 + code(T)
= (198 – 3*64) * 4 + 3
= 6*4 + 3 = 27

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 27

i = 3
Fingerprint(T) = (F(T) – T[2]*c) * 4 + T[6]
= (27– code(A)*64) * 4 + code(A)
= (27 – 0*64) * 4 + 0
= 27 * 4 = 108

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 108

i = 4
Fingerprint(T) = (F(T) – T[3]*c) * 4 + T[7]
= (108– code(C)*64) * 4 + code(G)
= (108 – 1*64) * 4 + 2
= 44 * 4 + 2 = 178

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 178

i = 5
Fingerprint(T) = (F(T) – T[4]*c) * 4 + T[8]
= (178– code(G)*64) * 4 + code(C)
= (178 – 2*64) * 4 + 1
= 50 * 4 + 1 = 201

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 201

i = 6
Fingerprint(T) = (F(T) – T[5]*c) * 4 + T[9]
= (201– code(T)*64) * 4 + code(T)
= (201– 3*64) * 4 + 3
= 9 * 4 + 3 = 39

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 39

i = 7
Fingerprint(T) = (F(T) – T[6]*c) * 4 + T[10]
= (39– code(A)*64) * 4 + code(A)
= (39– 0*64) * 4 + 0
= 39 * 4 + 0 = 156

F(P) != F(T) skip

Rabin-Karp

T =
TACGTAGCTAGTCGA
P = CTAG

A:0; C:1; G:2; T:3
Q = 997

$c = 64$

F (P) = 114
F (T) = 156

i = 7
Fingerprint(T) = (F(T) – T[7]*c) * 4 + T[11]
= (156– code(G)*64) * 4 + code(G)
= (156– 2*64) * 4 + 2
= 28 * 4 + 2 = 114

F(P) = F(T) CHECK!!!

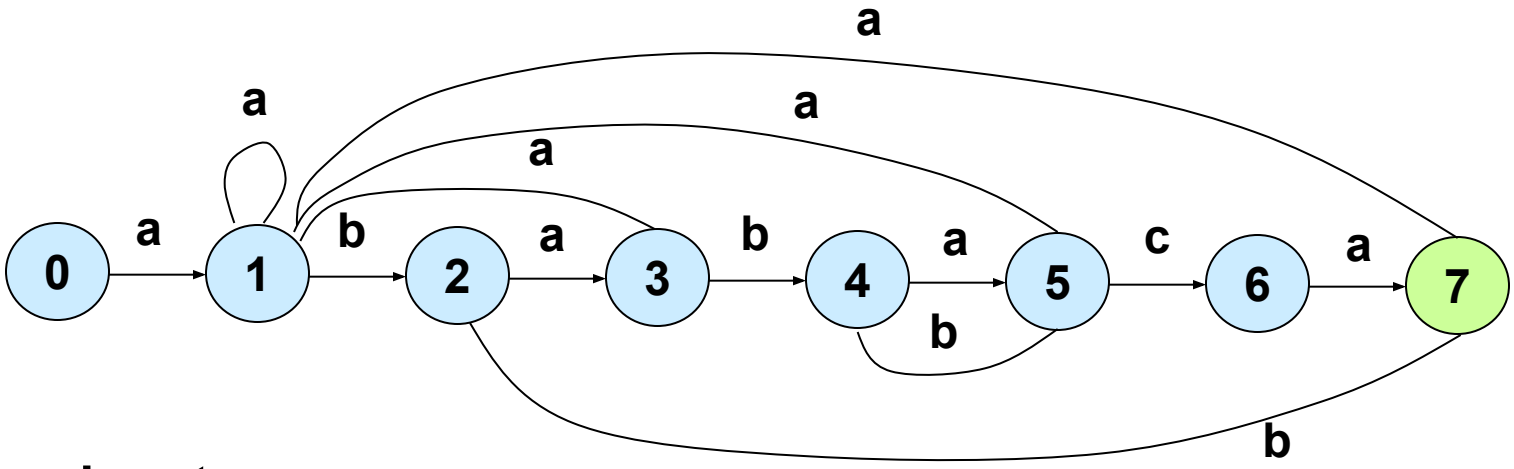
CTAG = CTAG

FINITE AUTOMATA

Searching in n comparisons

- The goal: each character of the text is compared only once!
- Problem with the naïve algorithm:
  - Forgets what was learned from a partial match!
  - Examples:
    - T = “Tweedledee and Tweedledum” and
      P = “Tweedledum”
    - T = “pappappappar” and P = “pappar”

Finite automaton search



input

state

	a	b	c	P
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 --	1	2	3	4	5	6	7	8	9	10	11
T[i] --	a	b	a	b	a	b	a	c	a	b	a
state φ(i) 0	1	2	3	4	5	4	5	6	7	2	3

Processing time takes Θ(n).
But have to first construct FA.
Main Issue: How to construct FA?

Need some Notation …

φ(w) = state FA ends up in after processing w.

Example: φ(abab) = 4.

σ(x) = max{k: P_k suf x}. Called the **suffix function**.

Examples: Let P = ab.

σ(ε) = 0

σ(ccaca) = 1

σ(ccab) = 2

Note: If |P| = m, then σ(x) = m indicates a match.

T: a b a b b a b b a c …

States: 0 1…...m……..m…….

match match

FA Construction

Given: P[1..m] Let Q = states = {0, 1, …, m}.

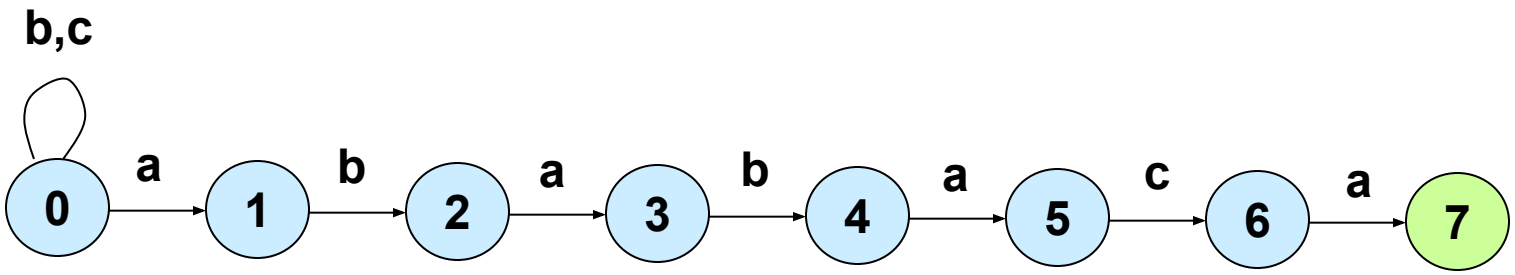
Define transition function δ as follows:

δ(q, a) = σ(P_qa) for each q and a.

Example: δ(5, b) = σ(P_5b) (P = ababaca)
= σ(ababab)
= 4

Intuition: Encountering a 'b' in state 5 means the current substring doesn't match. But, you know this substring ends with "abab" -- and this is the longest suffix that matches the beginning of P. Thus, we go to state 4 and continue processing "abab…" .

P=ababaca

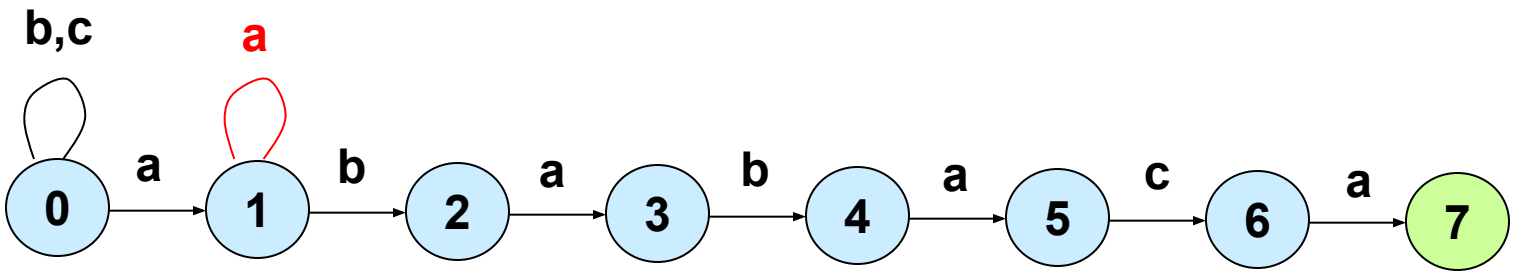


m=7; Q={0,1,2,3,4,5,6,7)

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

P=ababaca

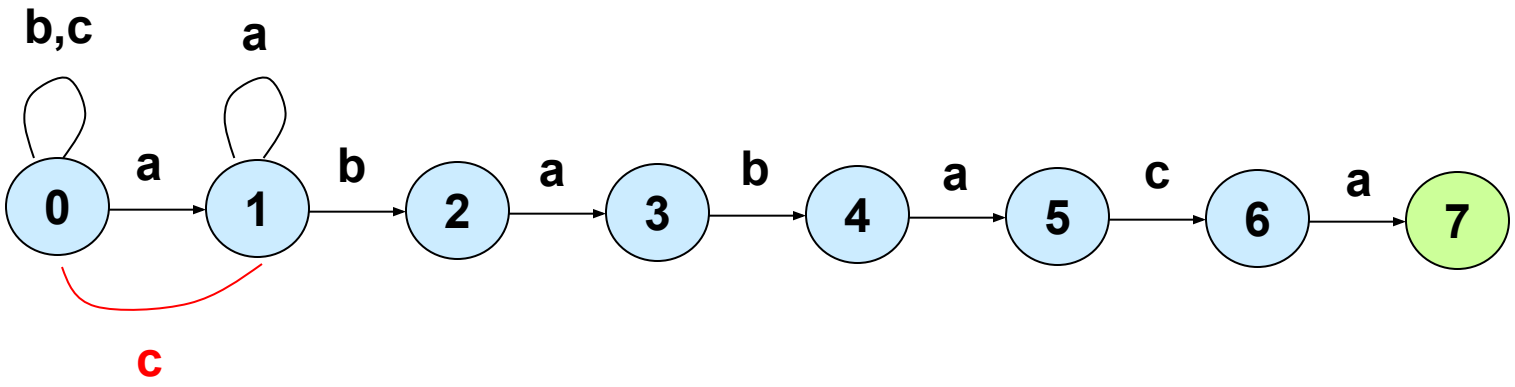


$\delta(1, a) = \sigma(P_1 a) = \sigma(aa) = \sigma(a) = 1$

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

P=ababaca



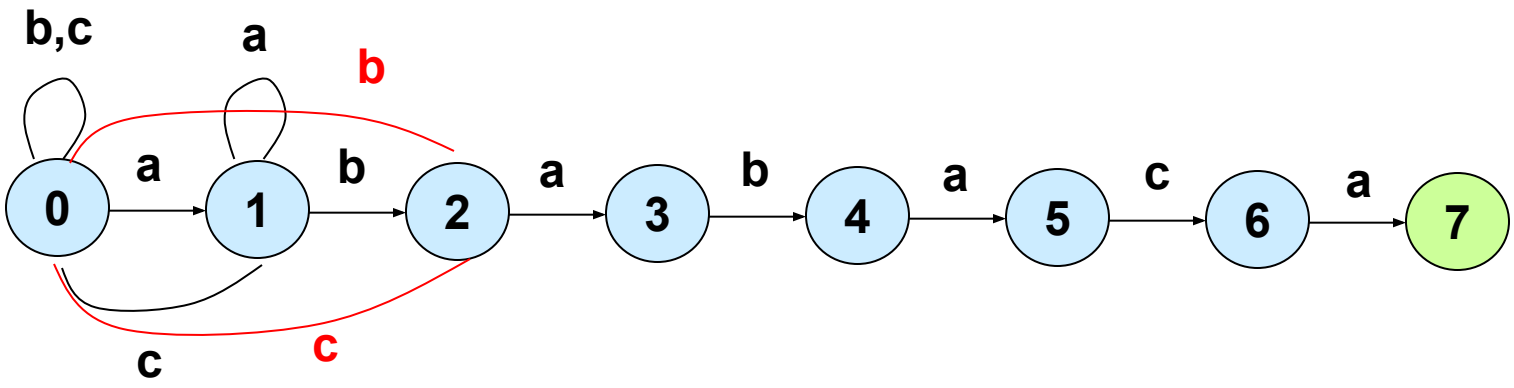
$\delta(1, a) = \sigma(P_1 a) = \sigma(aa) = \sigma(a) = 1$

$\delta(1, c) = \sigma(P_1 c) = \sigma(ac) = 0$

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

P=ababaca

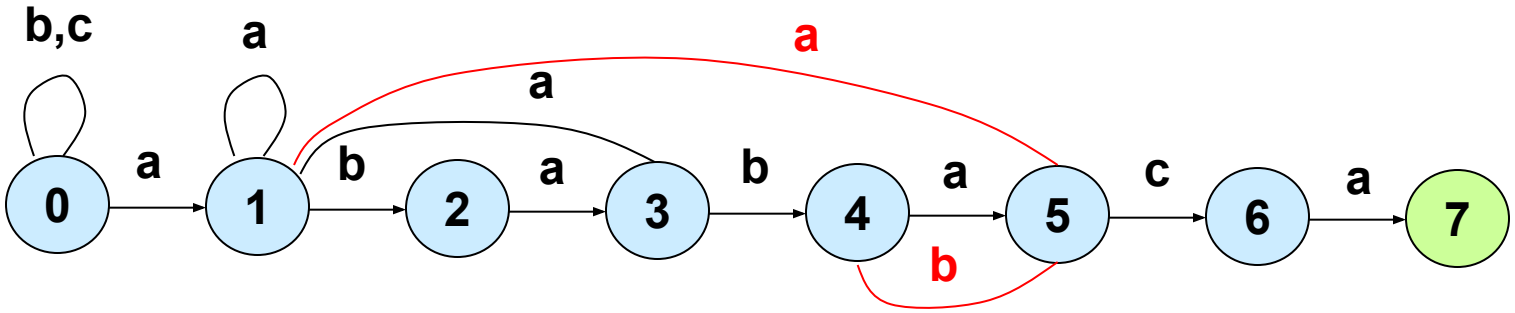


$\delta(2, b) = \sigma(P_2 b) = \sigma(abb) = 0$
$\delta(2, c) = \sigma(P_2 c) = \sigma(abc) = 0$

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

P=ababaca (fast forward & simplified)



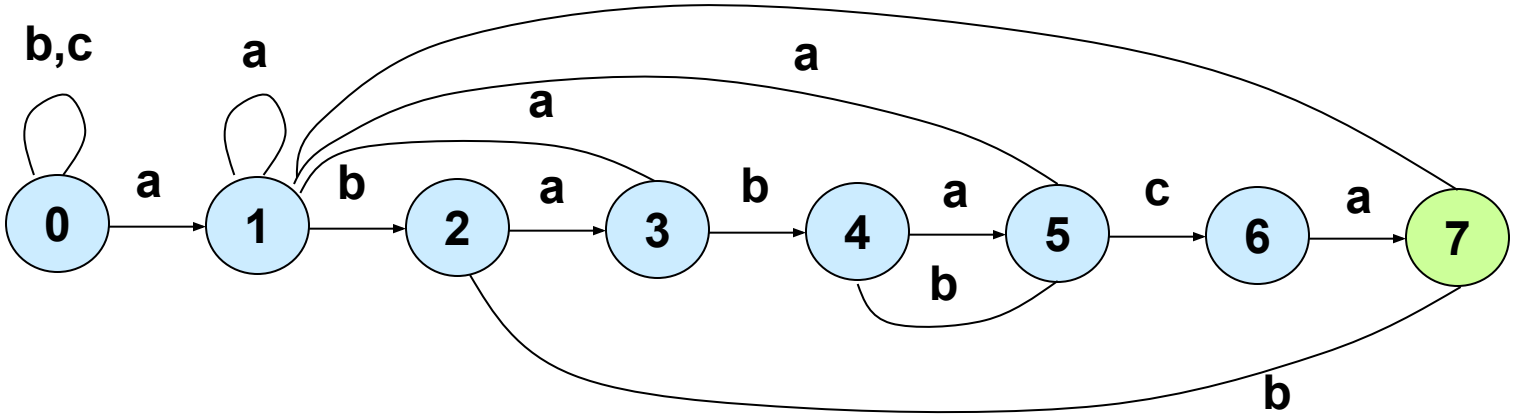
$\delta(5, a) = \sigma(P_5 a) = \sigma(ababaa) = \sigma(a) = 1$

$\delta(5, b) = \sigma(P_5 b) = \sigma(ababab) = \sigma(abab) = 4$

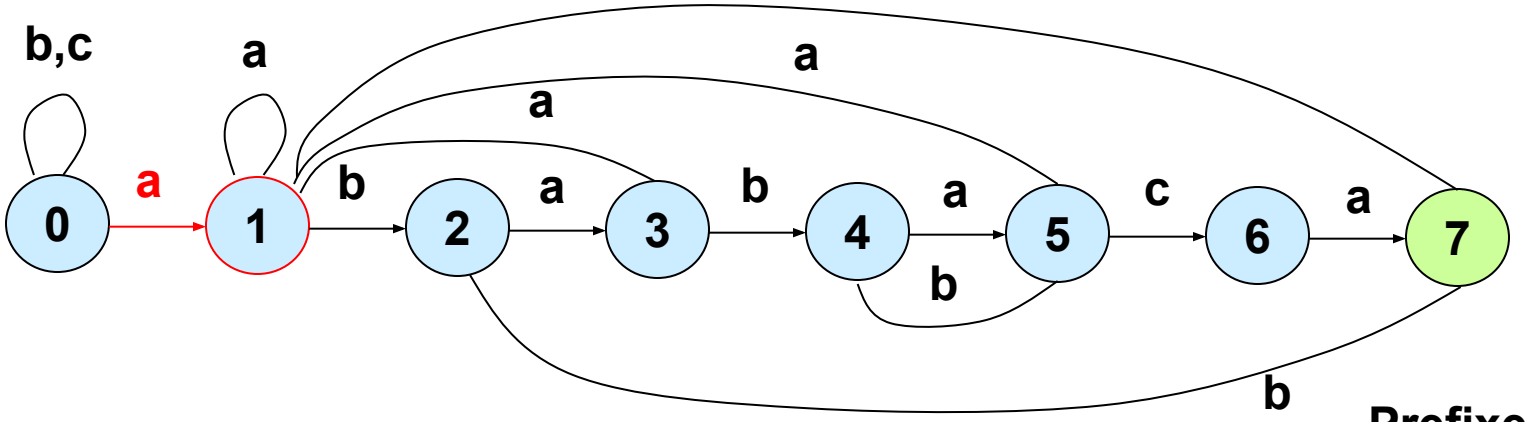
Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

P=ababaca (final, simplified)



Search

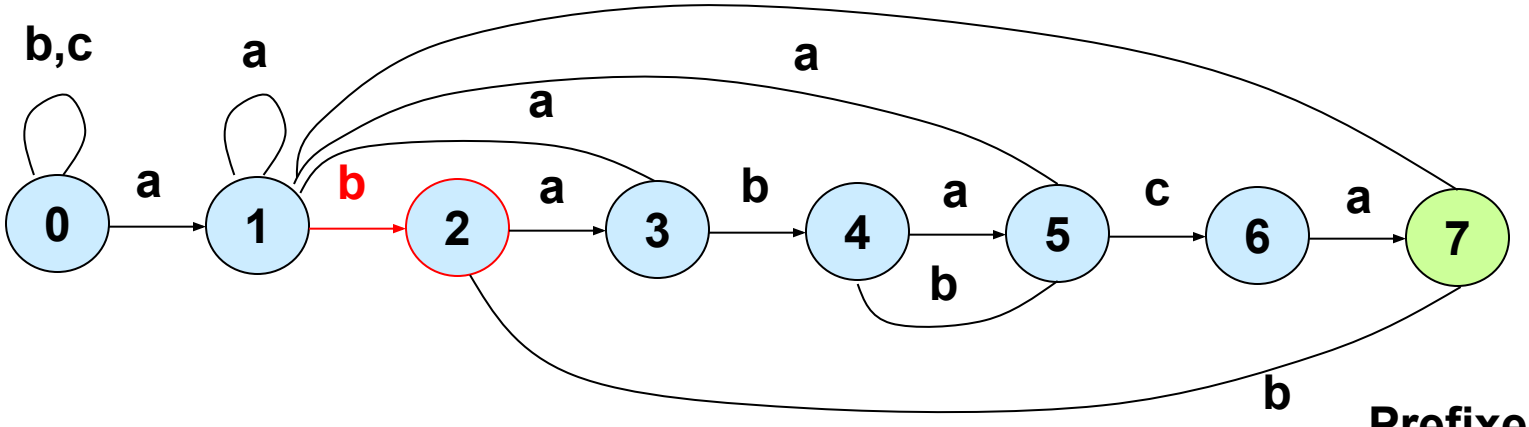


T= abababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

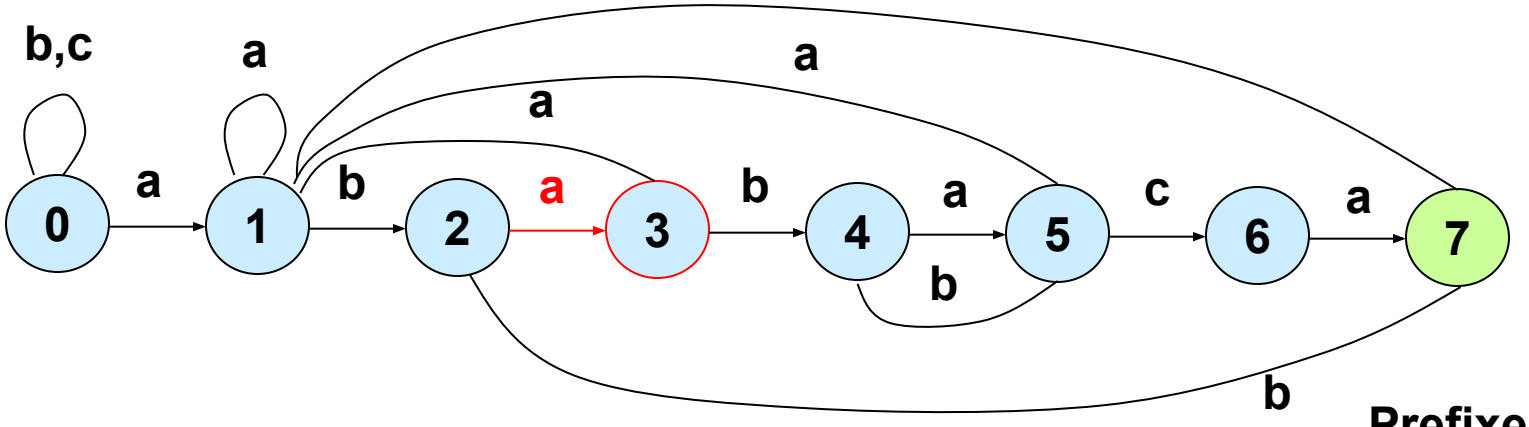


T= abababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

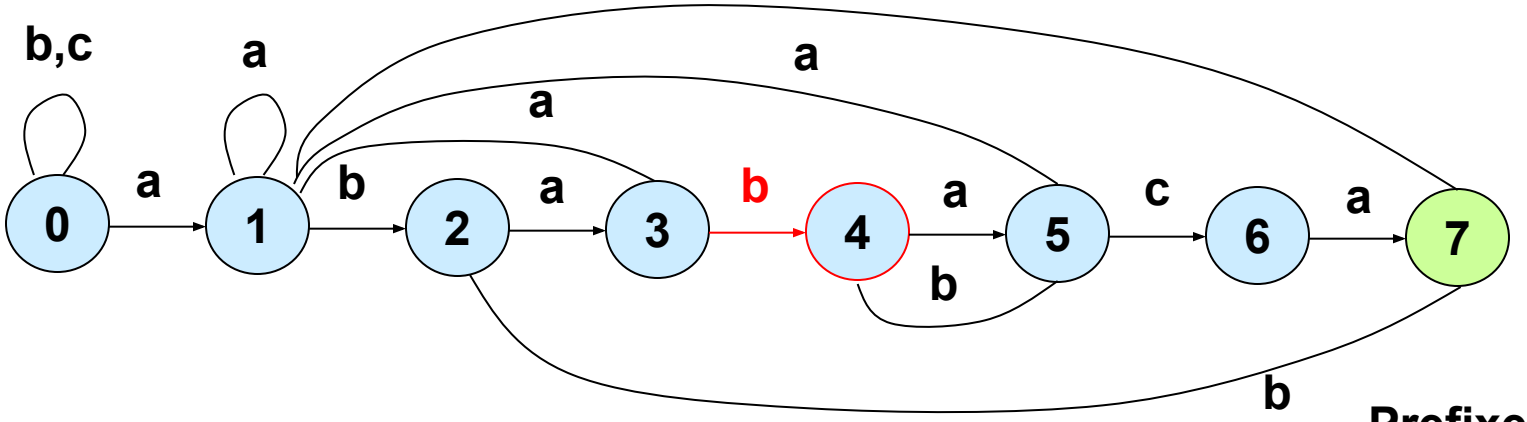


T= ababababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

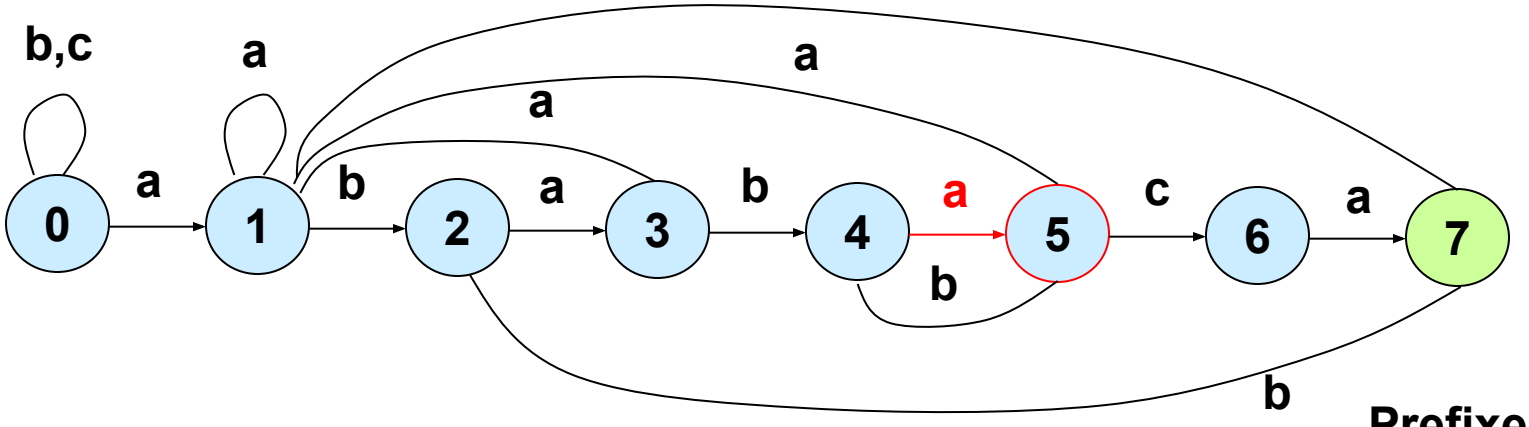


T= ababababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

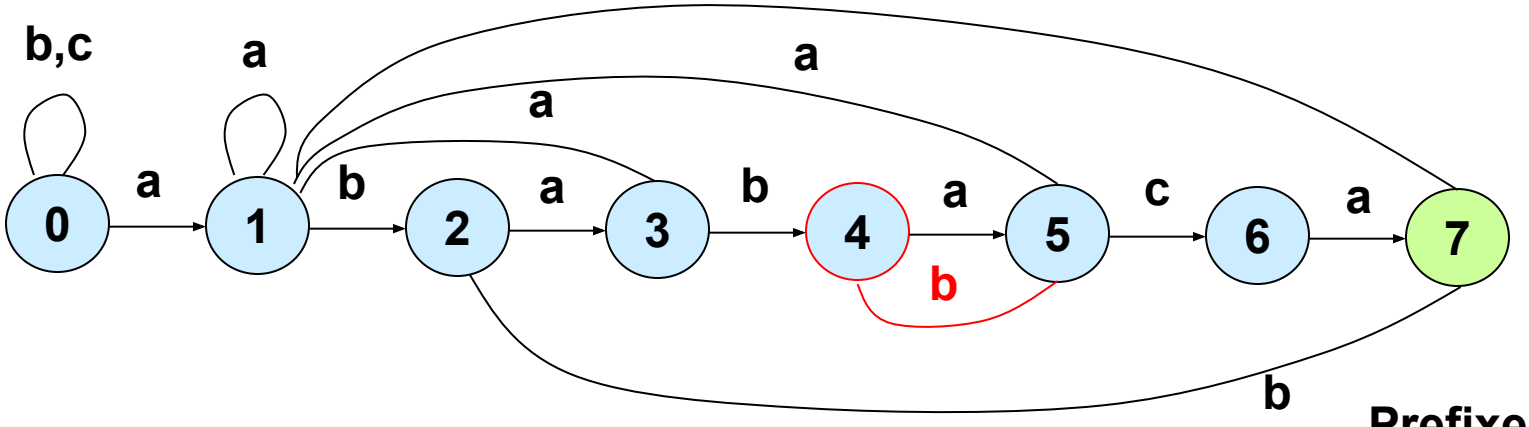


T= ababababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

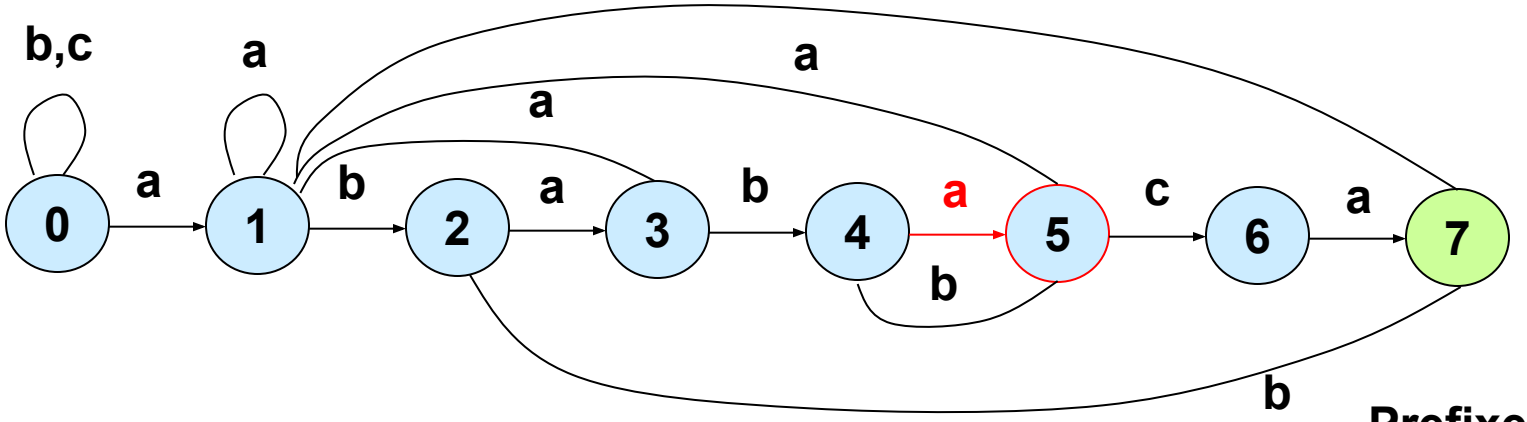


T= ababa**b**acaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

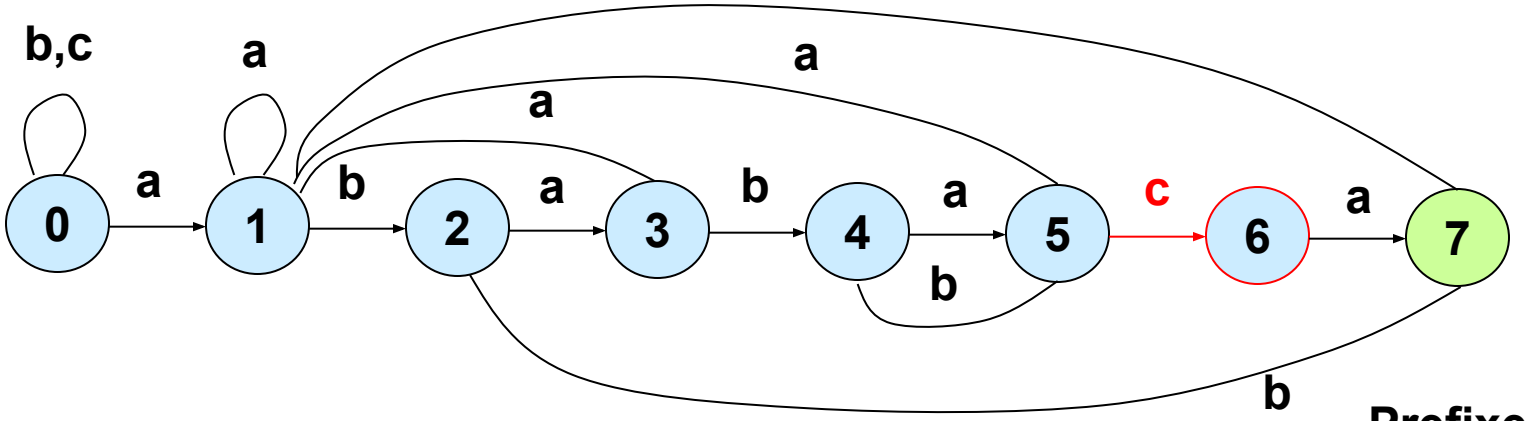


T= abababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search

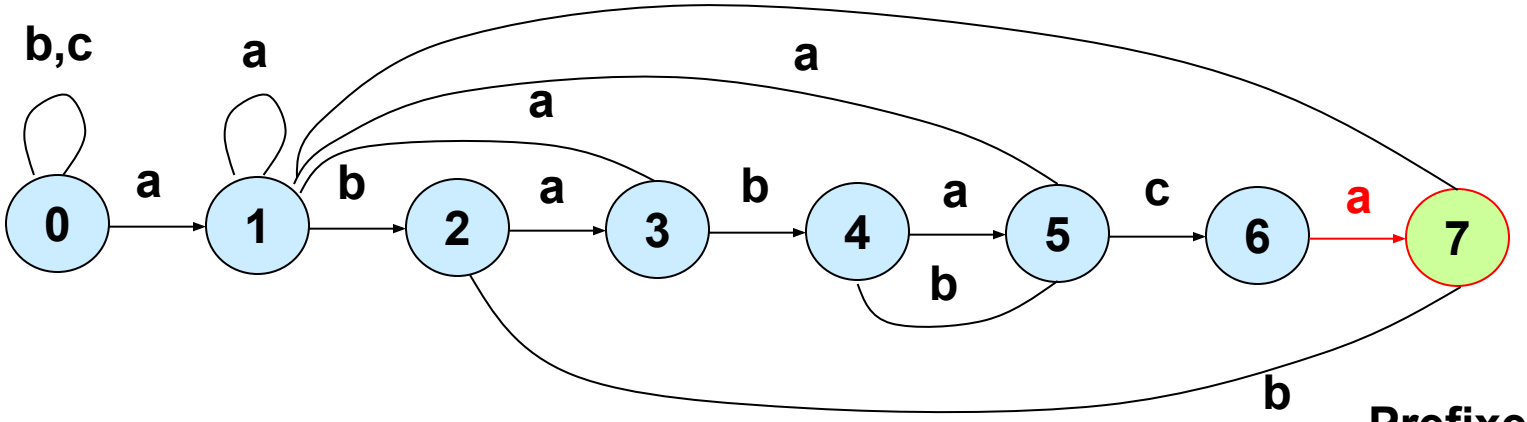


T= abababacaba

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Search



T= abababacaba

Accept state, we are done

Prefixes

a
ab
aba
abab
ababa
ababac
ababaca

Analysis of FA

- Searching: O(n) → good
- Preprocessing: O(m|Σ|) → bad
- Memory: O(m|Σ|) → bad

BITAP ALGORITHM: SHIFT/AND

The *Shift-And* Method

- Define M to be a binary n by m matrix such that:

  M(*i,j*) = *1* iff the first *i* characters of P exactly match the *i* characters of T ending at character *j.*

  M(*i,j*) = *1* iff P[*1* .. *i*] ≡ T[*j-i+1* .. *j*]

The *Shift-And* Method

- **Let T = california**
- **Let P = for**

M =

	1	2	3	4	5	6	7	8	9	m = 10
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0

- **M(*i*,*j*) = *1* iff the first *i* characters of P exactly match the *i* characters of T ending at character *j*.**

How to construct M

- We will construct M column by column.
- Two definitions:
- *Bit-Shift(j-1)* is the vector derived by *shifting* the vector for column *j-1* down by one and setting the first bit to *1*.
- Example:

$$\textbf{\textit{BitShift}}\begin{pmatrix}0\\1\\1\\0\\1\end{pmatrix}=\begin{pmatrix}1\\0\\1\\1\\0\end{pmatrix}$$

How to construct M

- We define the n-length binary vector U(*x*) for each character *x* in the alphabet. U(*x*) is set to *1* for the positions in P where character *x* appears.
- Example:

P = abaac $\boldsymbol{U(a)} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$ $\boldsymbol{U(b)} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ $\boldsymbol{U(c)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$

How to construct M

- **Initialize column 0 of M to all zeros**
- **For *j* > 1 column *j* is obtained by**

$$M(j) = BitShift(j-1) \wedge U(T(j))$$

An example $j = 1$

1 2 3 4 5 6 7 8 9 10

T = x a b x a b a a c a

1 2 3 4 5

P = a b a a c

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

$$\boldsymbol{U}(\boldsymbol{x}) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\boldsymbol{BitShift}(0) \,\&\, \boldsymbol{U}(\boldsymbol{T}(1)) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \,\&\, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

An example $j = 2$

1 2 3 4 5 6 7 8 9 10

T = x a b x a b a a c a

1 2 3 4 5

P = a b a a c

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

$$U(a) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$BitShift(1) \,\&\, U(T(2)) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \,\&\, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

An example $j = 3$

1 2 3 4 5 6 7 8 9 10

T = x a b x a b a a c a

1 2 3 4 5

P = a b a a c

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

$$\boldsymbol{U}(\boldsymbol{b}) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\boldsymbol{BitShift}(2) \,\&\, \boldsymbol{U}(\boldsymbol{T}(3)) = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \,\&\, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

An example $j = 8$

1 2 3 4 5 6 7 8 9 10

T = x a b x a b a a c a

1 2 3 4 5

P = a b a a c

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

$$U(a) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$BitShift(7) \,\&\, U(T(8)) = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \,\&\, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Correctness

- For $i > 1$, Entry M(i,j) = 1 iff

1) The first $i-1$ characters of P match the $i-1$ characters of T ending at character $j-1$.
2) Character P(i) ≡ T(j).

- 1) is true when M($i-1$,$j-1$) = 1.
- 2) is true when the i^{th} bit of U(T(j)) = 1.

- The algorithm computes the ***and*** of these two bits.

Correctness

1 2 3 4 5 6 7 8 9 10

T = x a b x a b a a c a

a b a a c

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

- M(4,8) = *1*, this is because a b a a is a prefix of P of length 4 that ends at position 8 in T.
- Condition 1) – We had a b a as a prefix of length 3 that ended at position 7 in T ↔ M(3,7) = 1.
- Condition 2) – The fourth bit of P is the eighth bit of T ↔ The fourth bit of U(T(8)) = 1.

How much did we pay?

- Formally the running time is $\Theta(mn)$.
- However, the method is very efficient if m is the size of a single or a few computer words.
- Furthermore only two columns of M are needed at any given time. Hence, the space used by the algorithm is $O(m)$ for $m=|P|$.

Extension to Bitap

2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali[†⋈] Gurpreet S. Kalsi[⋈] Zülal Bingöl[▽] Can Firtina[◇] Lavanya Subramanian[‡] Jeremie S. Kim[◇†]
Rachata Ausavarungnirun[⊙] Mohammed Alser[◇] Juan Gomez-Luna[◇] Amirali Boroumand[†] Anant Nori[⋈]
Allison Scibisz[†] Sreenivas Subramoney[⋈] Can Alkan[▽] Saugata Ghose[⋆†] Onur Mutlu[◇†▽]
[†]*Carnegie Mellon University* [⋈]*Processor Architecture Research Lab, Intel Labs* [▽]*Bilkent University* [◇]*ETH Zürich*
[‡]*Facebook* [⊙]*King Mongkut's University of Technology North Bangkok* [⋆]*University of Illinois at Urbana–Champaign*

- Supports edits (not exact matching)
- Supports backtrack
- Low-power, processing-in-memory design