String Matching

Find all occurrences of a pattern in a text

String Matching Problem:

Given text array T[1..n] and pattern array P[1..m] of characters from alphabet Σ, find all s such that T[s+1..s+m] = P[1..m], i.e., P occurs with shift s in T.

Example

r	o	w		r	o	w		r	o	w		y	o	u	r		b	o	a	t
										P	=	y	o							

$$s = 12$$

$$\Sigma = \{a, b, o, r, t, u, w, y\}$$

T: yoyoyoyo

P: yoyo

String Matching

- Simple problem with many applications
  - text editing
  - pattern recognition

- Algorithms
  - Naive O((n-m+1)m) worst case
  - Rabin and Karp O((n-m+1)m) worst case, but better on average
  - Finite Automaton O(n+m| Σ |)
  - Knuth-Morris-Pratt O(n+m)
  - Boyer and Moore O((n-m+1)m+| Σ |) worst case, but better (best overall) in practice

---

Naive String Matching

```
Naive(T, P)
   n = length(T)
   m = length(P)
   for s = 0 to n-m                                O(n-m+1)
      if P[1..m] = T[s+1..s+m]                         O(m)
      then print "Pattern occurs with shift" s
```

This algorithm takes O((n-m+1)m) time.

However, there is more information in a failed match:

T:	a	a	a	a	b	a	a	...
P:	a	a	a	a	a			

s = s + m

No need to consider ______________________

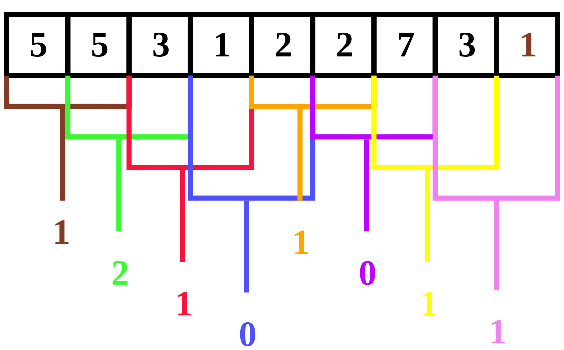
Rabin-Karp Algorithm

- Let characters be digits in radix-| Σ | notation.
- Choose a prime number q such that | Σ | q fits within a computer word to speed computations.

- Algorithm:
  Compute (P mod q)
  Compute (T[s+1, .., s+m] mod q) for s = 0 .. n-m
  Test against P only those sequences in T having the same (mod q) value
- (T[s+1, .., s+m] mod q) can be incrementally computed by subtracting the high-order digit, shifting, adding the low-order bit, all in modulo q arithmetic.

Example

$$\Sigma = \{0, 1, .., 9\}$$
$$P = 12,\ P \bmod 3 = 0$$
$$q = 3$$



Analysis

The Rabin-Karp algorithm takes $\Theta((n-m+1)m)$ time in the worst case.
O(n) + O(m(v + n/q)) average case, v = #valid shifts
If q $\geq$ m and v = O(1), then O(n+m).

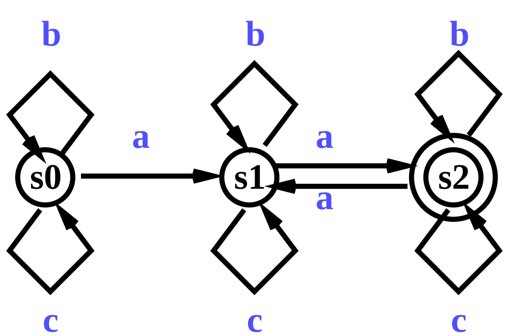
Finite Automata

A finite automata M = (Q, q_0, A, Σ, δ), where

- Q = set of states (s_i)
- q_0 = start state (s_0)
- A = set of accepting states
- Σ = input alphabet
- δ = transition function Q x$\Sigma \rightarrow$ Q

Example

Here is a finite automaton accepting strings with an even number of "a"s.
Σ = {a, b, c}.



$\delta(s_0, \text{a}) = s_1$
$\delta(s_0, \text{b}) = \delta(s_0, \text{c}) = s_0$
$\delta(s_1, \text{a}) = s_2$
$\delta(s_1, \text{b}) = \delta(s_1, \text{c}) = s_1$
$\delta(s_2, \text{a}) = s_1$
$\delta(s_2, \text{b}) = \delta(s_2, \text{c}) = s_2$

$A = \{s_2\}$

Consider input string w. If w ends at state $s \in A$, then the FA accepts w; otherwise, the FA rejects w.

Example: str = bccabaccaba

Accept

String Matching FA

1. Compute FA accepting P (m+1 states)
2. Run FA with input string T, printing shift whenever accepting state is reached.

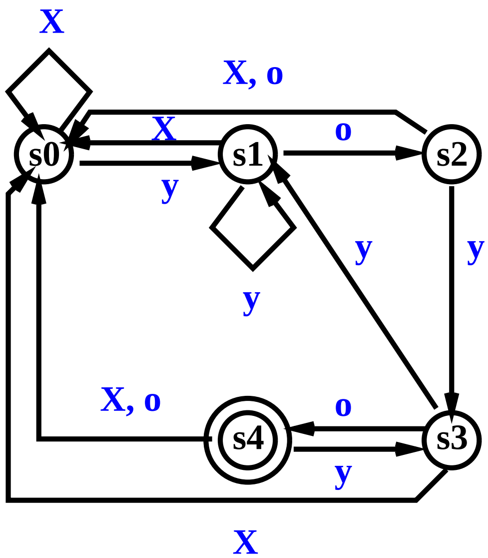
Example

P = yoyo, m=4

T = spin your yoyo

Σ = {i, n, o, p, r, s, u, y}

Let X = Σ - {y, o}



Analysis

Computing δ: O(m| Σ |)

FA-Matcher(T, δ, m)   ; O(n)
    n = length(T)
    s = s_0
    for i = 1 to n
        s = δ(s, T[i])
        if s = s_m
        then print "Pattern occurs with shift" (i-m)

This algorithm takes O(n + m| Σ |) time.

Knuth-Morris-Pratt Algorithm

- Utilize a prefix array π[1..m], where π[q] contains information to compute δ(q, a) for (a $\in \Sigma$), the pattern shift for a mismatch on P[q].

- π requires only O(m) time (as opposed to O(m$|\ \Sigma\ |$) for δ).

Prefix Array

Example

n	e	y	o	y	o	d	y	n	e	y	o
		y	o	y	o	s					

s = 2

How far can we shift P over and be assured of catching all matches?

Since we have matched up to yoyo and yo is a suffix of yoyo, then we can shift over by 2 and start testing at P[3].

Prefix Array

$\pi[q]$ answers the question:

If we have matched P[1..q] in T, but P[q+1] does not match, then what is the longest prefix of P, P[1..k], that is a suffix of P[1..q]?

We can then start matching again from P[k+1].

$$\pi[q] = \max\{k \mid k < q \text{ and } P[1..k] \text{ is a suffix of } P[1..q]\}$$

Example

$$P = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ y & o & y & o & s \end{array}$$

$$\pi = \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 2 & 0 \end{array}$$

Pseudocode

```
Compute-Prefix-Function(P)
    m = length(P)
    π[1] = 0                       ; k must be less than q
    k = 0
    for q = 2 to m                 ; O(m) amortized
        while k > 0 and P[k+1] ≠ P[q]
            k = π[k]
        if P[k+1] = P[q]
        then k = k + 1             ; prefix increased by one
        π[q] = k
    return π
```

Pseudocode

```
KMP-Matcher(T, P)
    n = length(T)
```

```
m = length(P)
π = Compute-Prefix-Function(P)          ; O(m) amortized
q = 0
for i = 1 to n                          ; O(n) amortized
    while q > 0 and P[q+1] ≠ T[i] ; where do we move to in P?
        q = π[q]
    if P[q+1] = T[i]                    ; matches so far
    then q = q + 1
    if q = m
    then print "Pattern occurs with shift" (i-m)
        q = π[q]
```

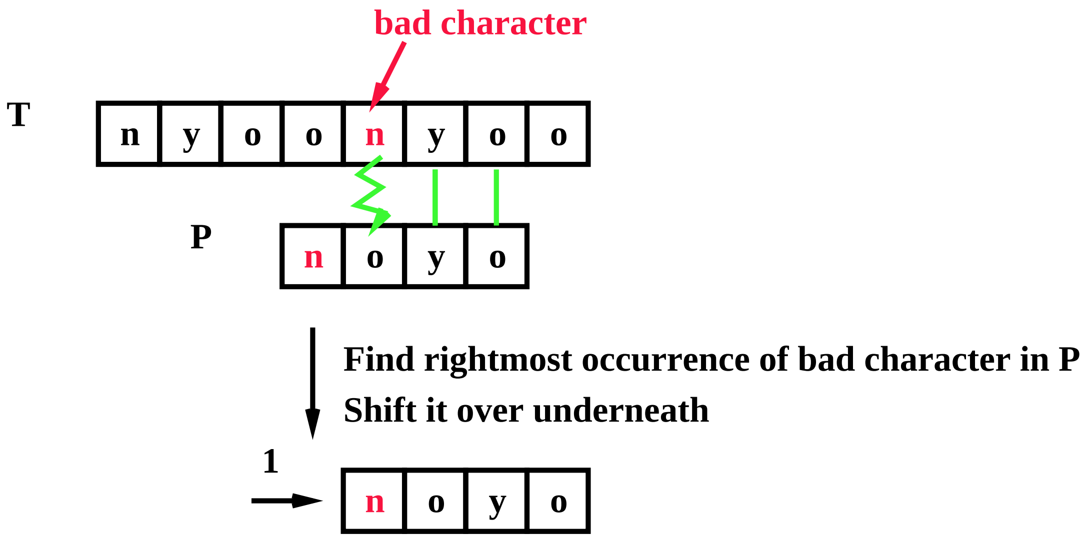
This algorithm takes ________ time

Boyer-Moore Algorithm

- Most efficient (on average) when P is long and Σ is large
- Matches pattern from right to left
- Utilizes two heuristics

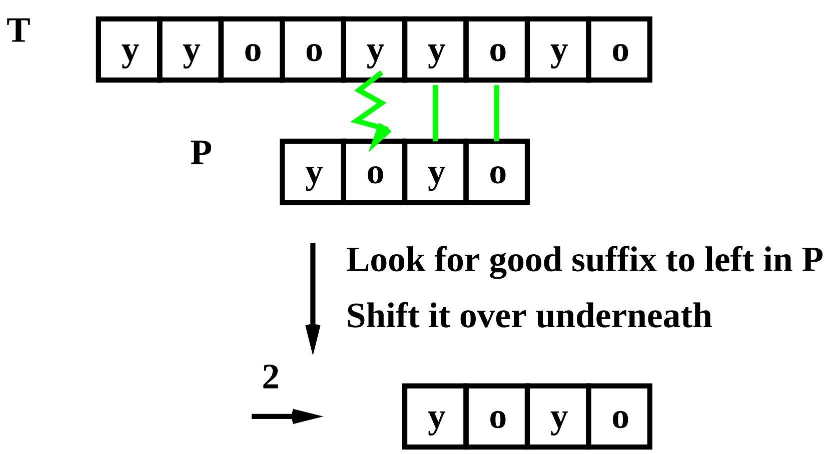
Bad Character Heuristic

Example



Good Suffix Heuristic

Example



Information For Bad Character Heuristic

```
Compute-Last-Occurrence(P, m Σ)
    foreach a ∈ Σ
        λ[a] = 0
    for j = 1 to m
        λ[P[j]] = j
    return λ
```

Running time: $O(|\Sigma| + m)$

If mismatch at P[j] $\neq$ T[s+j], then shift (j - λ[T[s+j]]).

Note: Shift could be negative, in which case ignore the shift value and use Good Suffix shift which always has a positive value.

Information for Good Suffix Heuristic

$$\gamma[j] = m - \max\{k \mid 0 \leq k < m \text{ and } P[j+1..m] \sqsupset P_k \text{ or } P_k \sqsupset P[j+1..m]\}$$

$\sqsupset$ means *suffix* (note: $x \sqsupset x$)

If match j+1..m and P[j] $\neq$ T[s+j], shift right $\geq \gamma$[j]

Examples

googoo

3	3	3	3	3	1	1

j = 0, $P_3 \sqsupset$ P[1..6]

googo

3	3	3	3	2	1

Pseudocode

```
Compute-Good-Suffix(P, m)
    π = Prefix(P)
    P' = reverse(P)
    π' = Prefix(P')
    for j = 0 to m                  ; O(m)
        γ[j] = m - π[m]
    for l = 1 to m
        j = m - π'[l]
        if γ[j] > l - π'[l]
        then γ[j] = l - π'[l]
    return γ
```

Example

$$m = 4$$
$$P = \text{yoyo},\ \pi = \_\_\_\_\_\_$$
$$P' = \text{oyoy},\ \pi' = \_\_\_\_\_\_$$
$$\gamma = \_\_\_\_\_\_\_$$
$$\gamma = \_\_\_\_\_\_\_$$

Boyer-Moore-Matcher

```
Boyer-Moore-Matcher(T, P, Σ)
    n = length(T)
    m = length(P)
    λ = Compute-Last-Occurrence(P, m, Σ)      ; O(| Σ | + m)
    γ = Compute-Good-Suffix(P, m)              ; O(m)
    s = 0
```

```
while s ≤ n-m                                    ; O(n-m+1)
    j = m
    while j > 0 and P[j] = T[s+j]                ; O(m)
        j = j - 1
    if j = 0
    then print "Pattern occurs with shift" s
        s = s + γ[0]
    else s = s + max(γ[j], j - λ[T[s+j]])
```

Close to naive

$O((n\text{-}m+1)m + |\Sigma|)$

Boyer-Moore-Matcher is actually best in practice

Example

T = soyoyo

P = yoyo

γ = ________

$\Sigma = \{o, s, y\}$

λ = ____

γ = ______________

$\xrightarrow{2}$ yoyo

Match

Applications