String Matching

Assume a text in an array $T[1,\ldots n]$ and a pattern $P[1..m]$, $m \leq n$.

The characters are drawn from alphabet set Σ.

Goal is to find occurences of P in T.

Example. $T = a\,a\,b\,c\,b\,a\,a\,b\,a\,a\,a$

$P = a\,a\,b$

$\Sigma = \{a, b, c\}$

P occurs at two places.

P occurs with shift s in T if $0 \leq s \leq n-m$ and $T[s+1, \ldots s+m] = P[1\ldots m]$, i.e. $T[s+j] = P[j]$ for $1 \leq j \leq m$.

Definitions for strings:

Σ^*: set of all strings possible with Σ

ϵ: empty string, length zero, belongs to Σ^*

$|x|$: length of string x.

xy: concatenation of strings x and y.

Prefix: $\omega \sqsubset x$ is a prefix of x if $x = \omega y$ for $y \in \Sigma^*$

Suffix: $\omega \sqsupset x$ is a suffix of x if $x = y\omega$ for $y \in \Sigma^*$

The empty string is both prefix and suffix of every string.

aab c d bba
prefix suffix

P_k: k-character prefix of $P[1...m]$, that is, $P_k = P[1...k]$.

T_k: k-character prefix of $T[1...n]$.

String matching: Find all shifts $s \in [0, n-m]$ so that $P \sqsupset T_{s+m}$

Checking equality "$x = y$": takes $O(t)$ time if t is the length of the longest string that is prefix of both x and y.

"aabcaab = aabaabc" takes (3+1)= 4 checks.

A straightforward algorithm

Check if $P[1 \ldots m] = T[s+1 \ldots s+m]$ for each $s \in [0, n-m]$.

```
Straight-Match (T, P)
  n := length [T]
  m := length [P]
  for s := 0 to n-m
      do if P[1...m] = T[s+1...s+m]
            then print "match with shift s".
```

Each check takes $O(m)$ time and there are ~~$O(n-m)$~~ $(n-m+1)$ checks. So, total time $O((n-m)m)$

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

a b a b

a b a b

a b a b

⋮

It should be obvious that starting from second position is redundant from looking at P. This is utilized for efficiency later

Rabin-Karp Algorithm

Here the strings are mapped to numbers which are matched instead. The algorithm has $\Theta(m)$ preprocessing time and $\Theta((n-m)m)$ running time. So, it is not better than the straightforward algorithm in the worst-case, but runs faster in practice.

Assume $\Sigma = \{0, 1, \ldots d-1\}$.
Each character is a digit in radix-d notation.

Example String 13456 in radix-10 is the number 13,456.

Let p be value of P in radix-10
t_s be value of $T[s+1, \ldots s+m]$ in radix-10

Then $p = t_s$ iff s is a valid shift, that is, P matches in T with shift s.

Compute p from P in $O(m)$ time

Compute all t_s from T in $O(n-m+1) = O(n-m)$ time

Check if $p = t_s$ for all $s \in [0, n-m]$.

This takes $O(m) + O(n-m) = O(n)$ time.

One can compute p by Horner's rule.

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \cdots + \cdots)\cdots).$$

t_0 can be computed similarly from T.

$$t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1].$$

Example. $m = 5$, $t_s = 31415$, $T[s+1] = 3$, $T[s+5+1] = 2$:

$$t_{s+1} = 10(31415 - 10000 \cdot 3) + 2$$
$$= 14152$$

So, after computing t_0 in $O(m)$ time, we can compute all $t_1, \ldots t_{n-m}$ in $O(n-m)$ time.

The difficulty with the previous scheme is that p and t_s can become too large to assume that each operation (arithmatic) is constant-time computation.

use modulo q numbers: p, t_s are all computed modulo q for some suitable q.

$$t_{s+1} = \left(d\left(t_s - T[s+1]\,h\right) + T[s+m+1]\right) \bmod q$$

where $h = d^{m-1} \pmod q$ ... [radix-d].

This solution keeps all numbers within a limited size $\le q-1$, but $p = t_s$ check is no more perfect, since $t_s \equiv p \pmod q$ does not imply $t_s = p$. However, if $t_s \not\equiv p \pmod q$, then $t_s \neq p$ for sure.

So, we can eliminate some invalid shifts quickly. But, if $t_s \equiv p \pmod q$, then we have to check further.

```
Rabin-Karp(T, P, d, q)
  n := length[T]
  m := length[P]
  h := d^(m-1) (mod q)
  p := 0
  t_0 := 0
  for i := 1 to m                                   } Preprocess
      do p := (dp + P[i]) mod q
         t_0 := (dt_0 + T[i]) mod q

  for s := 0 to n-m                                 } match
      do if p = t_s
  *      then if P[1...m] = T[s+1...s+m]
              then print "match with shifts"

  if s < n-m                                        } update t_s
      then t_{s+1} := (d(t_s - T[s+1]h)
                       + T[s+m+1]) mod q
```

Because of the check at *, the worst-case time is $O((n-m)m)$.

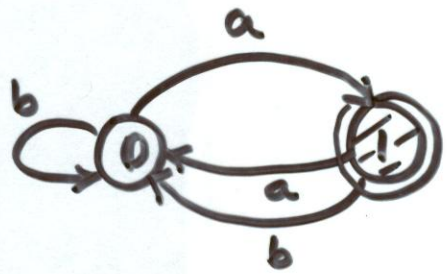
String Matching with FA

This method preprocesses the pattern P and builds a finite automaton. Then, matching needs $O(1)$ time per character in T. Thus, matching takes $O(n)$ time. But, preprocessing could be a little costly.

FA: $M = (Q, q_0, A, \Sigma, \delta)$

Q: finite states
$q_0 \in Q$: start state
$A \subseteq Q$: accepting states
Σ: input alphabet
$\delta : Q \times \Sigma \to Q$: a transition function.

State	a	b
0	1	0
1	0	0



If M consumes a string and ends up in a final state, M accepts the string. It rejects the string otherwise.

M induces a function $\phi : \Sigma^* \to Q$.

$$\phi(\epsilon) = q_0$$
$$\phi(wa) = \delta(\phi(w), a) \text{ for } w \in \Sigma^*, a \in \Sigma.$$

Automaton for P

Define a suffix function σ on P as follows:

$$\sigma(x) = \max\{k : P_k \sqsupset x\} \text{ for } x \in \Sigma^*.$$

$\sigma(x)$ is the length of the longest prefix of P that is a suffix of x.

Ex. $P = abab$

$\sigma(ab) = 2$

$\sigma(abaa) = 1$

$\sigma(abbb) = 0$

$\sigma(x) = m$ iff $P \sqsupset x$

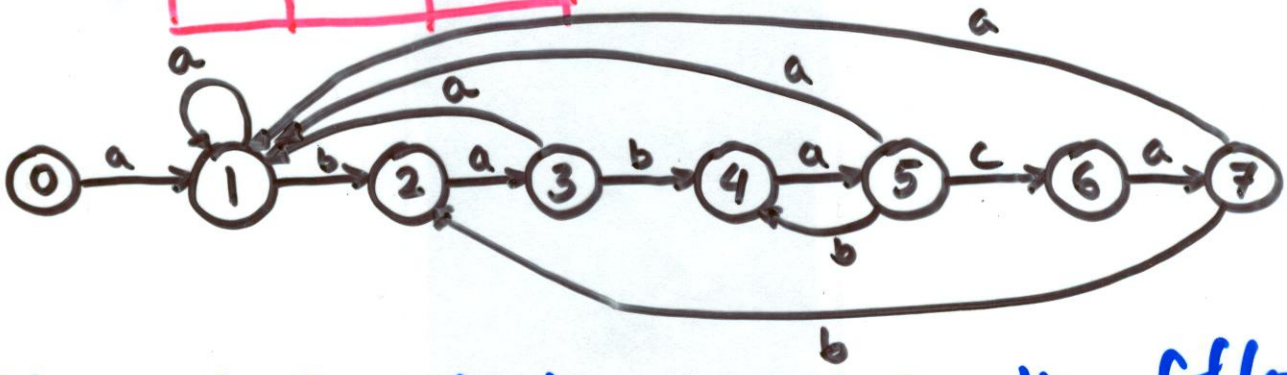
For a pattern $P[1 \ldots m]$ we define the automaton as:

- $Q: \{0, 1, \ldots, m\}$, $q_0 = \{0\}$, $A = \{m\}$.
- $\delta(q, a) = \sigma(P_q a)$.

Ex. P = ababaca

State	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:	1	2	3	4	5	6	7	8	9	10
T[i]:	a	b	a	b	a	b	a	c	a	b
ϕ(T[i]):	1	2	3	4	5	4	5	6	7	2



M is designed to maintain the following invariant: $\phi(T_i) = \sigma(T_i)$.

Now, assuming that M has been built for a P, we can write the algorithm for matching P in T.

```
Finite-Atomaton-match ( T, δ, m)
  n := length (T)
  q := 0
  for i := 1 to n
      do q := δ(q, T[i])
        if q = m
            then print "match with shift i-m"
```

Lemma 1. For any x and $a \in \Sigma$, we have

$$\sigma(xa) \leq \sigma(x) + 1.$$

Proof. Let $r = \sigma(xa)$. If $r = 0$, then $r \leq \sigma(x)+1$ trivially true. So, assume $r \neq 0$.

$P_r \sqsupset xa$ by def. of σ.

$\Downarrow$

$P_{r-1} \sqsupset x$

$\Downarrow$

$r-1 \leq \sigma(x)$.

Lemma 2. If $q = \sigma(x)$, then $\sigma(xa) = \sigma(P_q a)$.

Proof. $P_q \sqsupset x$ by def. of σ

① ... $P_q a \sqsupset xa$ straightforward

② $r = \sigma(xa) \leq q+1$ by Lemma 1

③ ... $P_r \sqsupset xa$ by def. $r = \sigma(xa)$.

④ ... $|P_r| \leq |P_q a|$ by ②

1, 3, 4 $\Rightarrow$ $P_r \sqsupset P_q a \Rightarrow r \leq \sigma(P_q a)$

$\Rightarrow \sigma(xa) \leq \sigma(P_q a)$

We also have $\sigma(P_q a) \leq \sigma(xa)$ by ①

Therefore, $\sigma(xa) = \sigma(P_q a)$.

Theorem If ϕ is the final-state function, then

$\phi(T_i) = \sigma(T_i) \quad \forall i \in [0, n]$.

Proof. By induction on i.

for $i = 0$, trivially true since

$T_0 = \epsilon \Rightarrow \phi(T_0) = 0 = \sigma(T_0)$.

Assume $\phi(T_i) = \sigma(T_i)$ and prove

$$\phi(T_{i+1}) = \sigma(T_{i+1}).$$

Let $q = \phi(T_i)$ and $a = T[i+1]$.

$$\begin{aligned}
\phi(T_{i+1}) &= \phi(T_i a) \\
&= \delta(\phi(T_i), a) \\
&= \delta(q, a) \\
&= \sigma(P_q a) \quad \text{(Def. of } \delta\text{)} \\
&= \sigma(T_i a) \quad \text{(Lemma 2)} \\
&= \sigma(T_{i+1})
\end{aligned}$$

By the Theorem, if M enters q, then q is the largest value s.t. $P_q \sqsupset T_i$. Thus, $q = m$ iff. P has occurred just currently. So, the finite-automaton algorithm is correct.

Transition function

```
Trans Function (P, Σ)
    m := length (P)
    for q := 0 to m                                   } O(m|Σ|)
        for each a ∈ Σ
            k := min(m+1, q+2)
            repeat k := k-1                           ← O(m)
            until P_k ⊐ P_q a                         ← O(m)
            δ(q,a) := k
    return δ
```

Total complexity: $O(m^3 |\Sigma|)$.

It can be improved to $O(m|\Sigma|)$.

Then FA approach takes $O(n)$ matching time with $O(m|\Sigma|)$ preprocessing time.