Rabin-Karp algorithm

→ Consider a character as a number in Radix system

eg: English alphabet as in radix 26

→ Pick up each m-length "number" starting from shift 0 to (n-m)

→ Ex:

T = g t g a t c a g a t c a c t in radix 4

(g/0, t/1, a/2, c/3)

becomes

g t g = 2 1 2 in base 4 = 32 + 4 + 2 (in decimal)

→ Then do the comparisions with p (number wise)

Note

Advantage :- Calculating strings can reuse old results

T[s]

4 3 5 9 4

Ex: 4 3 5 9 3 5 9 2

⇒ 3592 = (4359 − 4 * 1000) * 10 + 2

general formula

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

in radix d

where t_s = corresponding no for the substring

T[s+1 .. s+m] m = size of p

T[1 .. 1+m]

$0 \le s \le n-m$ (1)

problem :- in case each number is too large for comparision

soln :- Hash, use modular arithmetic,

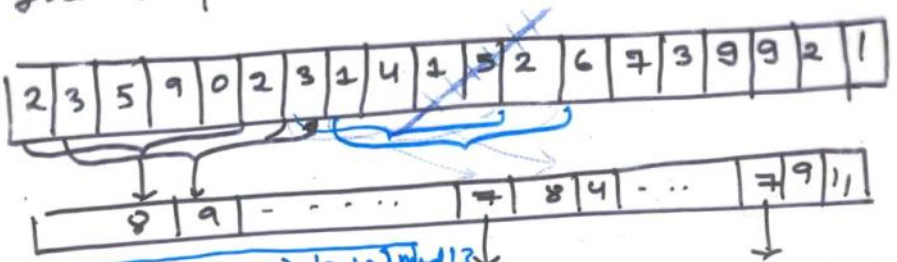
New recurrence formula

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

q = prime and $h = d^{m-1} \bmod q$.

→ comparision is not perfect, may have spurious hit

→ so we need a naïve matching when the comparision succeeds in modulo math



$14152 \equiv [(31415 - 3 \cdot 10000) \cdot 10 + 2] \bmod 13$
$\equiv [7 - 3 \cdot 3] \cdot 10 + 2 \bmod 13$
$\equiv 8 \bmod 13$

Valid Match

spurious hit

Pattern: 31415

$= 7$

use Horner's rule :-

$$a_0 + a_1 x + a_2 x^2 + \cdots = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots)))$$

$$\Rightarrow p = P[m] + 10(P[m-1] + 10(P[m-2] + 10(P[m-3] + \cdots)))$$

(2)

Rabin-Karp algorithm

Input: Text string T, pattern string to search for P.
radix to be used d (= $|\Sigma|$, for alphabet Σ)
q prime no.

→ **Algo:-**

```
n = length(T)        m = length(P)
h = d^(m-1) mod q
p = 0 ; t0 = 0
for i ← 1 to m                    // Preprocessing
p = (d * p + P[i]) mod q          // use Horner's rule.
t0 = (d * t0 + T[i]) mod q
end
for s ← 0 to n-m
  if (p == ts) then
    if (P[1..m] == T[s+1,..s+m]) then
      print s ;
  if (s < n-m) then
    ts+1 = (d(ts - h * T[s+1]) + T[s+m+1]) mod q
end.
```

Sorting in linear time

Counting sort (A, B, k)

```
1. for i ← 1 to k                                   ||  Θ(k)
       do C[i] ← 0
2. for j ← 1 to length [A]                          ||  Θ(n)
       do C[A[j]] ← C[A[j]] + 1.
3. for i ← 2 to k                                   ||  Θ(k)
       do C[i] ← C[i] + C[i-1]
4. for j ← length[A] downto 1.                      ||  Θ(n)
       do B[C[A[j]]] ← A[j]
          C[A[j]] ← C[A[j]] - 1.
```

→ Counting sort is an stable algorithm

Example:

A: 3 6 4 1 3 4 1 4

Example 2

A: 2 5 3 0 2 3 0 3

initial C array would be

0	1	2	3	4	5
2	2	4	7	7	8

(1)