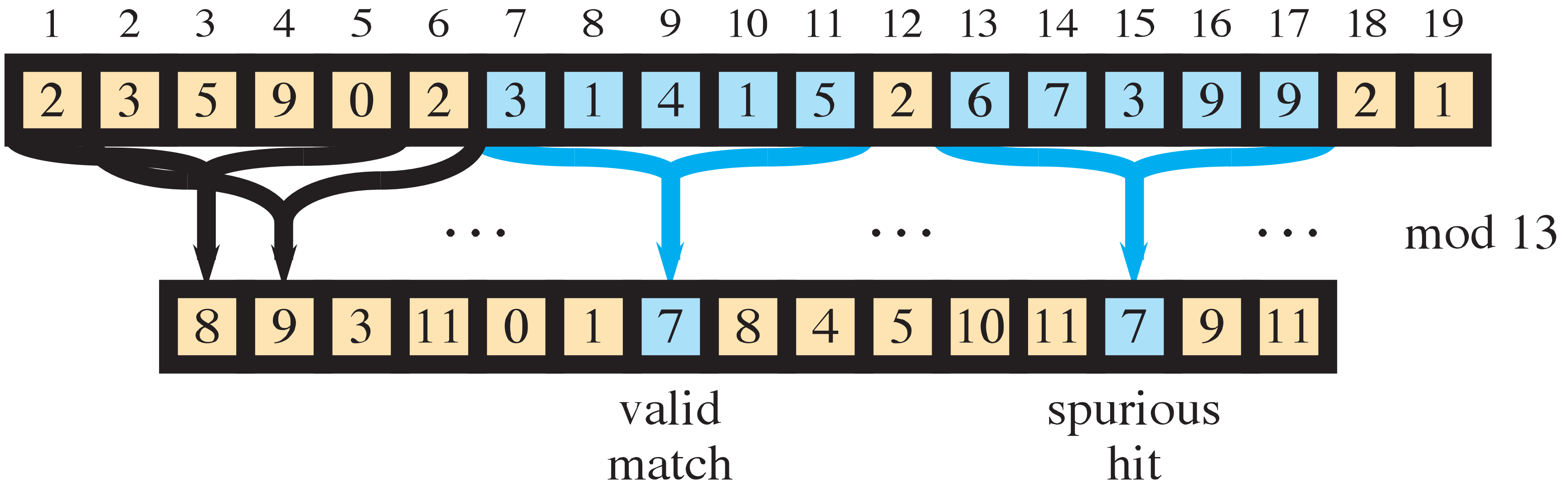
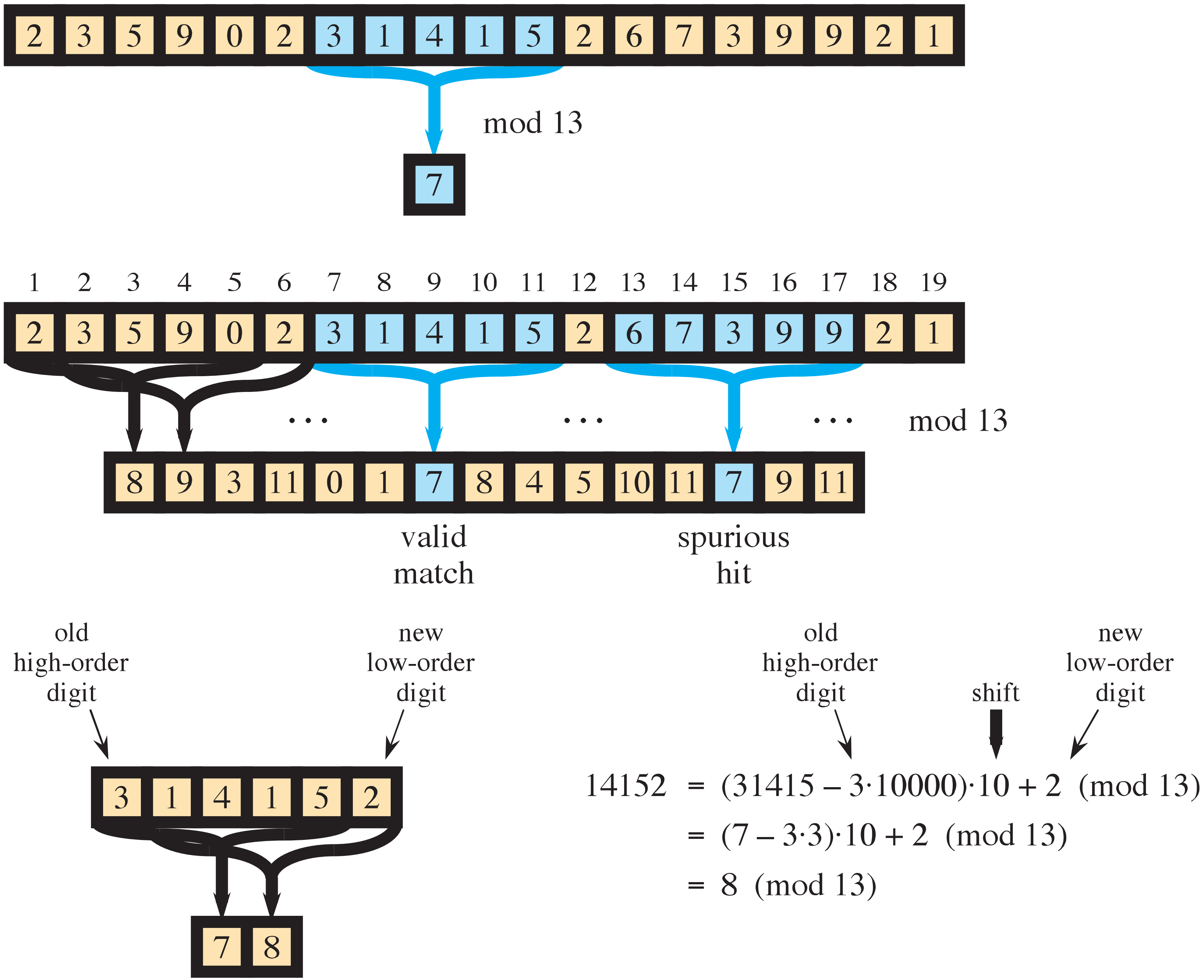
Rabin-Karp Algorithm
for
Pattern Searching

Key ideas

1. A hash function returns an integer (hash value) based on a given input.
2. Use hash values of strings to decide before performing brute-force pattern matching.
3. For the first window, compute hash value directly.
4. For subsequent windows, use a rolling hash function to increase efficiency.

   (Rolling hash functions compute hash value of next window based on the current value.)




2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
mod 13
7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
mod 13
8 9 3 11 0 1 7 8 4 5 10 11 7 9 11
valid match
spurious hit
old high-order digit
new low-order digit
3 1 4 1 5 2
7 8
old high-order digit
shift
new low-order digit
14152 = (31415 – 3·10000)·10 + 2 (mod 13)
= (7 – 3·3)·10 + 2 (mod 13)
= 8 (mod 13)

Overview of Rabin Karp

```
RabinKarp(string text[1..n], string pattern[1..m])
    if length(text) < length(pattern)
        return not possible

    hpat_txt = direct_hash(pattern[1..m])
    hval_txt = direct_hash(text[1..m])
    for i = 1 to n-m+1
        if hval_txt = hval_pat
            if text[i..i+m-1] == pattern[1..m]
                return i
        hval_txt = rolling_hash(text[i..i+m], hval_txt)

    return not found
```

Modular arithmetic for positive integers

a and b are said to be congruent modulo n, represented as (mod n), if their remainders of a and b when divided by n are equal.

- $5 \equiv 7 \pmod{2}$
- $52 \equiv 24 \pmod{7}$
- $31415 \equiv 67399 \pmod{13}$

Properties of addition in modular arithmetic for positive integers a, b, c, d, k and n:

1. If $a + b = c$, then $a \pmod{n} + b \pmod{n} \equiv c \pmod{n}$
2. If $a \equiv b \pmod{n}$, then $a + k \equiv b + k \pmod{n}$ $a + k \equiv b + k \pmod{n}$ for any integer k
3. If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$ $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$
4. If $a \equiv b \pmod{n}$, then $-a \equiv -b \pmod{n}$

Properties of multiplication in modular arithmetic for positive integers a, b and n:

1. If $a \cdot b = c$, then $a \pmod{n} \cdot b \pmod{n} \equiv c \pmod{n}$
2. If $a \equiv b \pmod{n}$, then $ka \equiv kb \pmod{n}$ for any integer k
3. If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $ac \equiv bd \pmod{n}$

Eg: $15 * 23 \pmod{7} \equiv 15 \pmod{7} . 23 \pmod{7} \pmod{7}$	$20 + 30 = 50$
$345 \pmod{7} \equiv 1 . 2 \pmod{7}$	$20 \pmod{7} + 30 \pmod{7} \equiv 50 \pmod{7}$
$2 \equiv 2 \pmod{7}$	$6 + 2 \equiv 1$
	$1 \equiv 1 \pmod{7}$

Horner's rule for polynomials (nested form)

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

$$p(x) = a_0 + x\left(a_1 + x\left(a_2 + x\left(a_3 + \cdots + x(a_{n-1} + x\,a_n)\cdots\right)\right)\right).$$

$5x^4 + 2x^3 - 3x^2 + x - 7$ at $x = 3$

$= (((5x + 2)\,x - 3)\,x + 1)\,x - 7$

$= (((5\,(3) + 2)\,x - 3)\,x + 1)\,x - 7$
$= ((17x - 3)\,x + 1)\,x - 7$

$= ((17\,(3) - 3)\,x + 1)\,x - 7$
$= (48x + 1)\,x - 7$

$= (48\,(3) + 1)\,x - 7$
$= 145x - 7$

$= 145\,(3) - 7$
$= 428$

```
base = 256 # Using ASCII values
mod = 1000000007 # A large prime number
```

```
pow(base, exp, mod=None)
   = base**exp          (if mod == None)
   = base**exp % mod    (if mod != None)
```

```python
base = 256 # Using ASCII values
mod = 1000000007 # A large prime number

hval = 0 # Find hash value directly for the first window using Horner’s rule
for i in range(win_size) :
    hval = (hval * base + ord(t[i])) % mod

yield hval
```

```
pow(base, exp, mod=None)
   = base**exp          (if mod == None)
   = base**exp % mod    (if mod != None)
```

```python
base = 256 # Using ASCII values
mod = 1000000007 # A large prime number

hval = 0 # Find hash value directly for the first window using Horner's rule
for i in range(win_size) :
    hval = (hval * base + ord(t[i])) % mod # ord() returns ASCII value

yield hval

# For subsequent windows use rolling hash
for i in range(win_size, len(t)) :
    # Remove the contribution of the outgoing character
    hval = (hval - ord(t[i - win_size]) * pow(base, win_size - 1, mod)) % mod

    # Add the contribution of the incoming character
    hval = (hval * base + ord(t[i])) % mod

    yield hval
```

```
pow(base, exp, mod=None)
  = base**exp         (if mod == None)
  = base**exp % mod   (if mod != None)
```

Overview of Rabin Karp

```
RabinKarp(string text[1..n], string pattern[1..m])
    if length(text) < length(pattern)
        return not possible

    hpat_txt = direct_hash(pattern[1..m])
    hval_txt = direct_hash(text[1..m])
    for i = 1 to n-m+1
        if hval_txt = hval_pat
            if text[i..i+m-1] == pattern[1..m]
                return i
        hval_txt = rolling_hash(text[i..i+m], hval_txt)

    return not found
```

Complexities?