

String matching

```
>>>
>>> import re
>>> r=re.compile(r"regexes\s(are|do)\s?(n[o']t)?\s(fun|boring)", re.I)
>>> def is_match(x): return x is not None
>>> is_match(re.match(r, "Regexes are fun!!!"))
True
>>> is_match(re.match(r, "Regexes are not fun!!!"))
True
>>> is_match(re.match(r, "Regexes aren't boring!!!"))
True
>>> is_match(re.match(r, "Obviously, regexes are boring. "))
False
>>> is_match(re.search(r, "Obviously, regexes are boring. "))
True
```

Announcements

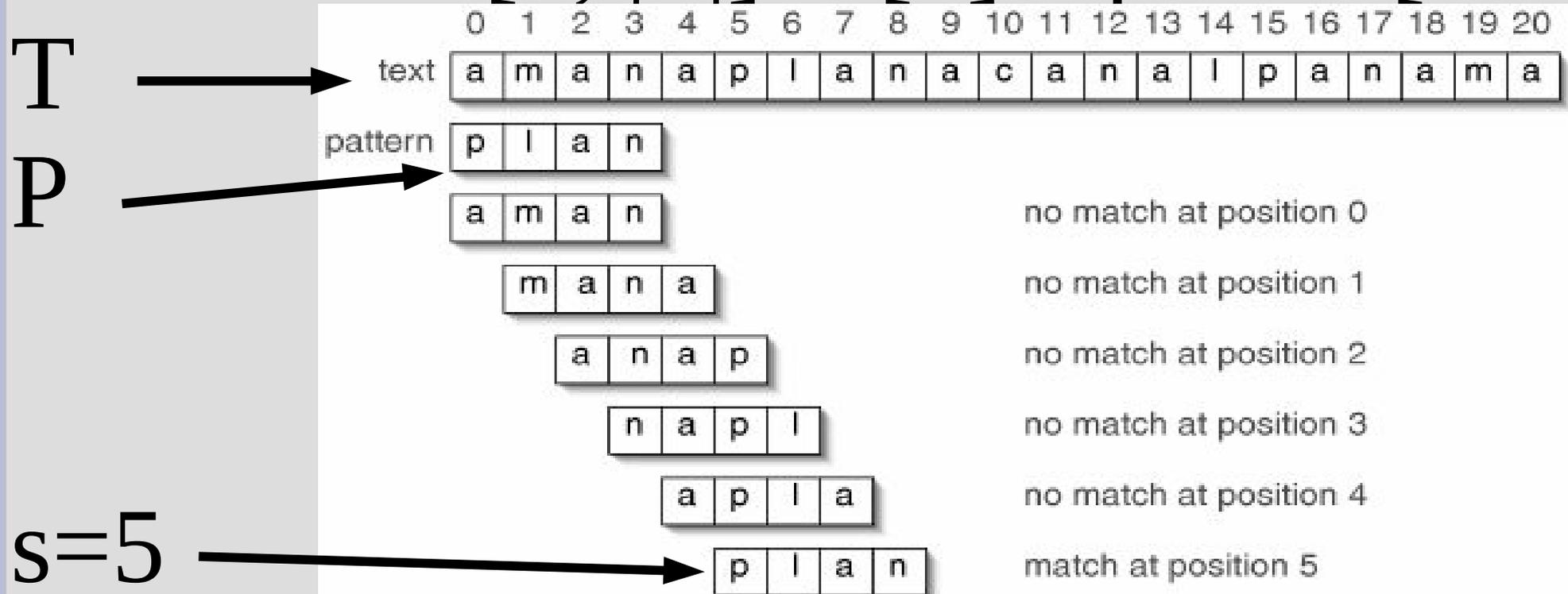
Programming assignment 1 posted
- need to submit a .sh file

The .sh file should just contain
what you need to type to
compile and run your program
from the terminal

String matching

Some pattern/string P occurs with shift s in text/string T if:

for all k in $[1, |P|]$: $P[k]$ equals $T[s+k]$



String matching

Both the pattern, P , and text, T , come from the same finite alphabet, Σ .

empty string (“”) = ε

w is a prefix of $x = w [x$, means exists y s.t. $wy = x$ (also implies $|w| \leq |x|$)

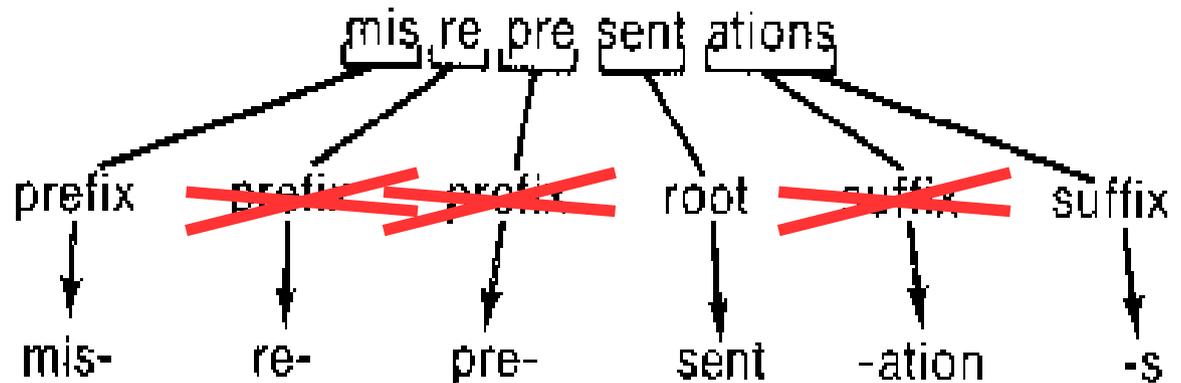
$(w] x = w$ is a suffix of x)

Prefix

w prefix of x means: all the first letters of x are w

X → "bread"
prefixes of x → b , br , bre , brea
suffixes of x → read , ead , ad , d

not
english!



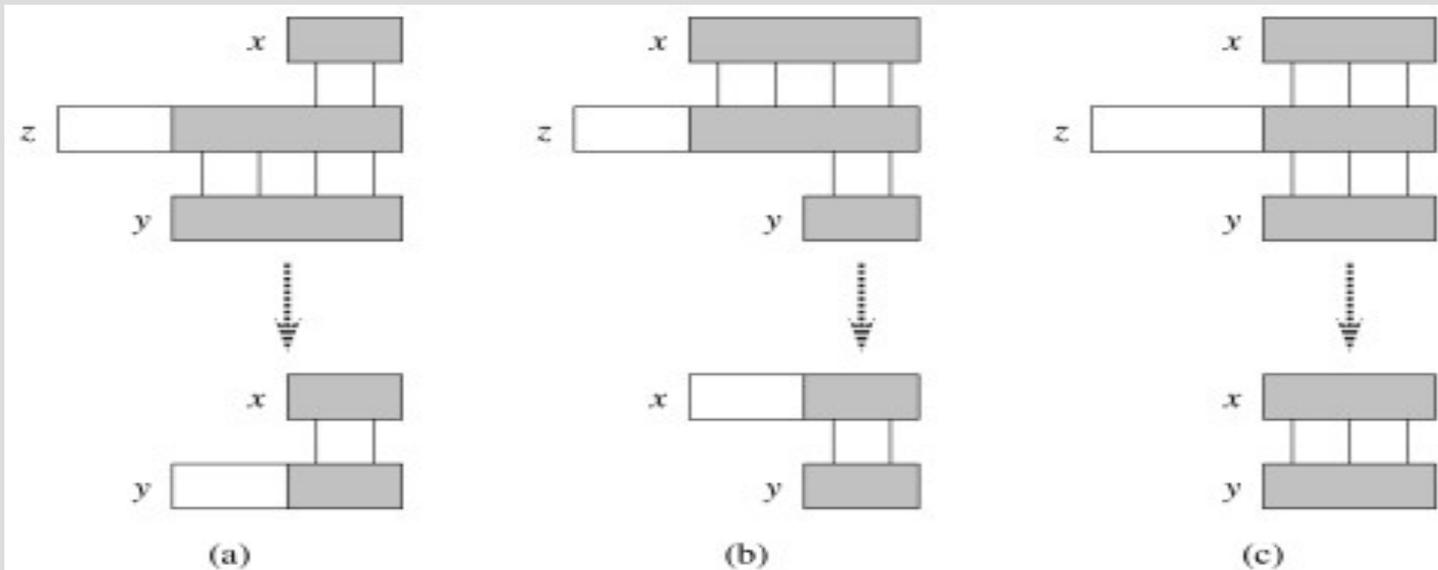
Suffix

If $x \supset z$ and $y \supset z$, then:

(a) If $|x| \leq |y|$, $x \supset y$

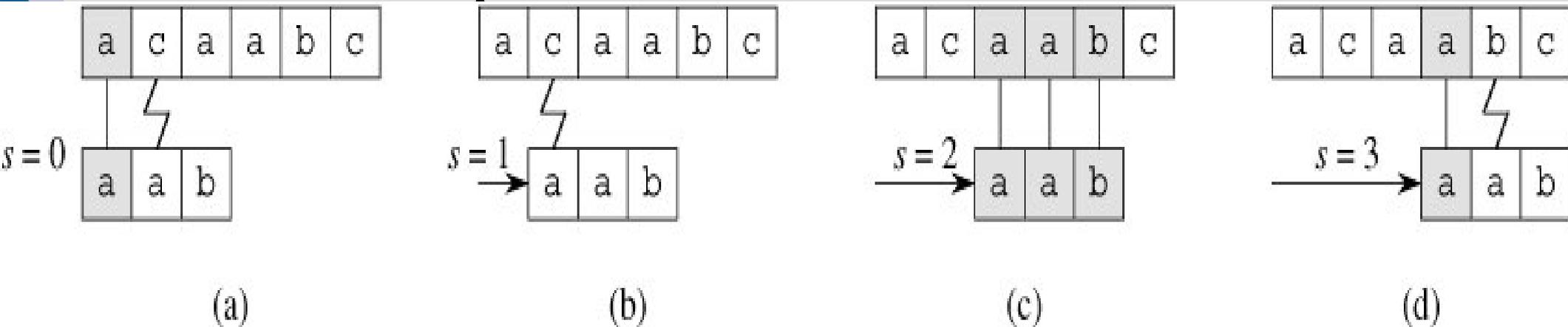
(b) If $|y| \leq |x|$, $y \supset x$

(c) If $|x| = |y|$, $x = y$



Dumb matching

Dumb way to find all shifts of P in T?
Check all possible shifts!

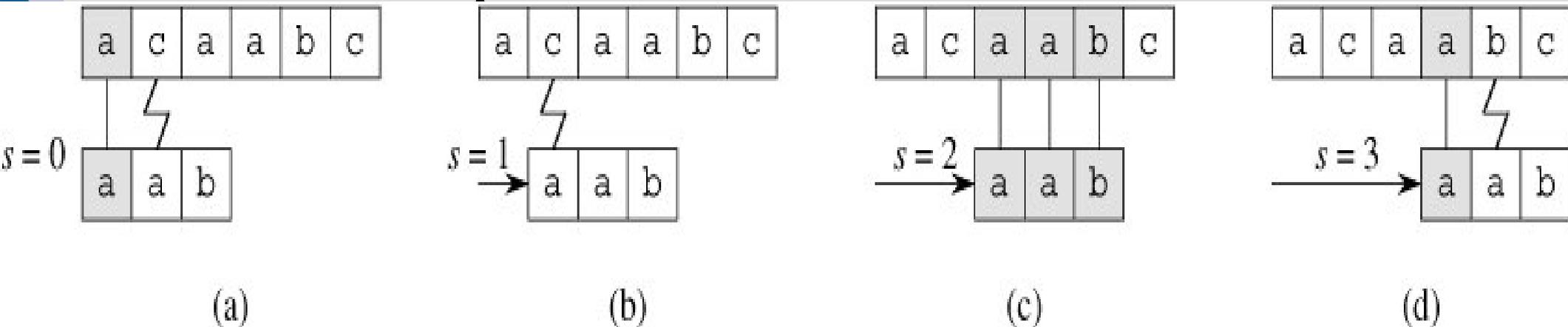


(see: naiveStringMatcher.py)

Run time?

Dumb matching

Dumb way to find all shifts of P in T?
Check all possible shifts!



(see: naiveStringMatcher.py)

Run time?

$O(|P| |T|)$

Rabin-Karp algorithm

A better way is to treat the pattern as a single numeric number, instead of a sequence of letters

So if $P = \{1, 2, 6\}$ treat it as 126 and check for that value in T

Rabin-Karp algorithm

The benefit is that it takes a(n almost) constant time to get the each number in T by the following:

(Let $t_s = T[s, s+1, \dots, s+|P|]$)

$$t_{s+1} = d(t_s - T[s+1]h) + T[s+|P|+1]$$

where $d = |\Sigma|$, $h = d^{|P|-1}$

Rabin-Karp algorithm

Example: $\Sigma = \{0, 1, \dots, 9\}$, $|\Sigma| = 10$

$T = \{1, 2, 6, 4, 7, 2\}$

$P = \{6, 4, 7\}$

$t_0 = 126$

$t_1 = 10(126 - T[0+1]10^{3-1}) + T[0+|P|+1]$

$t_1 = 10(126 - 100) + T[0+3+1]$

$t_1 = 264$

Rabin-Karp algorithm

This is a constant amount of work if the numbers are small...

So we make them small!
(using modulus/remainder)

Any problems?

Rabin-Karp algorithm

This is a constant amount of work if the numbers are small...

So we make them small!
(using modulus/remainder)

Any problems?

$x \bmod q = y \bmod q$ does not mean $x = y$

Hash functions



One way functions

Modulus is a one way function, thus computing the modulus is easy but recovering the original number is hard/impossible

$127 \% 5 = 2$, or $127 \bmod 5 = 2 \bmod 5$

However if we want to solve $x \% 5 = 2$, all we can say is $x = 2 + 5k$ or some k

One way functions

Other one way functions?

One way functions

Other one way functions?

- multiplication
- hashing

Multiplication is famous, as it is easy:

$$200 * 50 = 10,000$$

... yet factoring is hard:

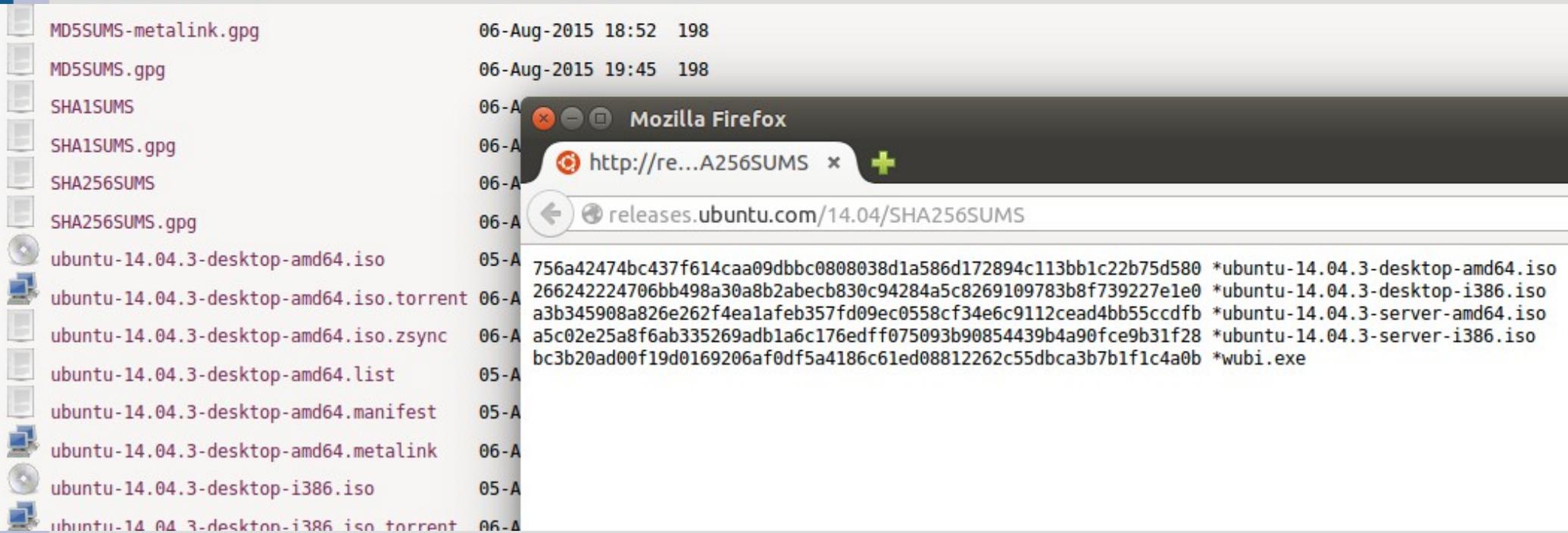
$$132773 = 31 * 4283 \text{ (what alg?)}$$

One way functions

Hashing is another commonly used function for security/verification, as...

- fast (low computation)
- low collision chance
- cannot easily produce a specific hash

One way functions



The image shows a terminal window with a list of files and their corresponding hashes and dates. A Firefox browser window is overlaid on the terminal, displaying the Ubuntu releases page for SHA256SUMS.

File Name	Date	Time	Size
MD5SUMS-metalink.gpg	06-Aug-2015	18:52	198
MD5SUMS.gpg	06-Aug-2015	19:45	198
SHA1SUMS	06-A		
SHA1SUMS.gpg	06-A		
SHA256SUMS	06-A		
SHA256SUMS.gpg	06-A		
ubuntu-14.04.3-desktop-amd64.iso	05-A		
ubuntu-14.04.3-desktop-amd64.iso.torrent	06-A		
ubuntu-14.04.3-desktop-amd64.iso.zsync	06-A		
ubuntu-14.04.3-desktop-amd64.list	05-A		
ubuntu-14.04.3-desktop-amd64.manifest	05-A		
ubuntu-14.04.3-desktop-amd64.metalink	06-A		
ubuntu-14.04.3-desktop-i386.iso	05-A		
ubuntu-14.04.3-desktop-i386.iso.torrent	06-A		

File Name	Hash	File Name
*ubuntu-14.04.3-desktop-amd64.iso	756a42474bc437f614caa09dbbc0808038d1a586d172894c113bb1c22b75d580	*ubuntu-14.04.3-desktop-amd64.iso
*ubuntu-14.04.3-desktop-i386.iso	266242224706bb498a30a8b2abecb830c94284a5c8269109783b8f739227e1e0	*ubuntu-14.04.3-desktop-i386.iso
*ubuntu-14.04.3-server-amd64.iso	a3b345908a826e262f4ea1afeb357fd09ec0558cf34e6c9112cead4bb55ccdfb	*ubuntu-14.04.3-server-amd64.iso
*ubuntu-14.04.3-server-i386.iso	a5c02e25a8f6ab335269adb1a6c176edff075093b90854439b4a90fce9b31f28	*ubuntu-14.04.3-server-i386.iso
*wubi.exe	bc3b20ad00f19d0169206af0df5a4186c61ed08812262c55dbca3b7b1f1c4a0b	*wubi.exe

Hash functions



Rabin-Karp algorithm

Larger q (for mod):

- larger numbers = more computation
- less frequent errors

There are trade-offs, but we often pick $q > |P|$ but not $q \gg |P|$

Pick a prime number as q

Rabin-Karp algorithm

Kabin-Karp-Matcher($T, P, |\Sigma|, q,$)

$d = |\Sigma|$, $h = d^{|P|-1} \bmod q$, $p = 0$, $t_0 = 0$

for $i = 1$ to $|P|$ // “preprocessing”

$p = (dp + P[i]) \bmod q$ // for P

$t_0 = (dt_0 + T[i]) \bmod q$ // for T

for $s = 0$ to $|T| - |P|$

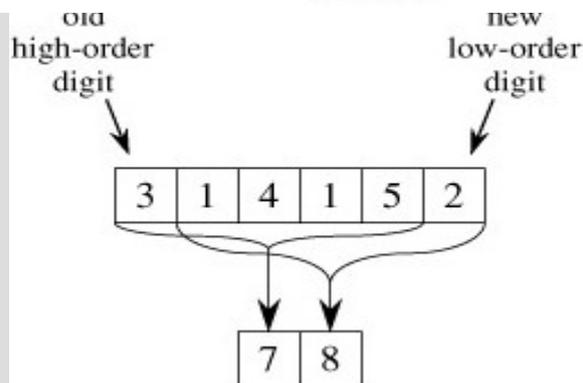
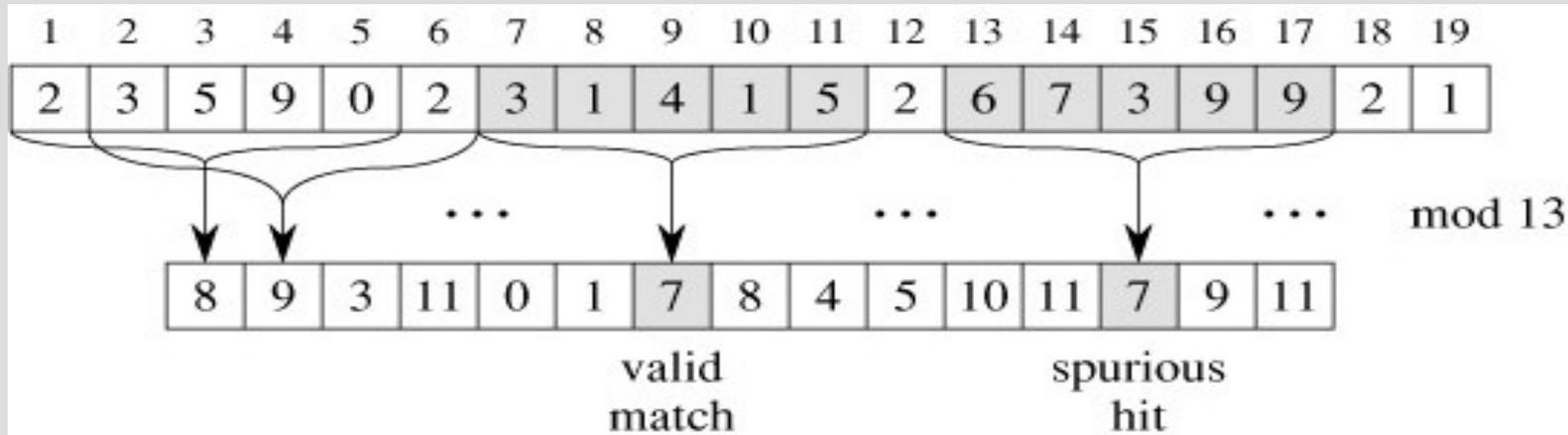
if $p == t_s$, check brute-force match at s

if $s < |T| - |P|$ then compute t_{s+1}

Rabin-Karp algorithm

To compute t_{s+1} :

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



$$\begin{aligned}
 14152 &\equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \\
 &\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13} \\
 &\equiv 8 \pmod{13}
 \end{aligned}$$

Rabin-Karp algorithm

Example: $T = \{1, 2, 5, 3, 5, 2, 6, 3\}$
 $P = \{2, 5\}$, $q = 5$, assume base 10

Rabin-Karp algorithm

Example: $T = \{1, 2, 5, 3, 5, 2, 6, 3\}$

$P = \{2, 5\}$, $q = 5$, assume base 10

$P = 25 \bmod 5 = 0$, $t_0 = 12 \bmod 5 = 2$

$t_{i+1} = 10 * (t_i - T[i+1] * 10) + T[i+|P|+1] \% q$

$t_1 = 25 \bmod 5 = 0$, true match!

$t_2 = 53 \bmod 5 = 3$,

$t_3 = 35 \bmod 5 = 0$, false match

Rabin-Karp algorithm

$$T = \{1, 2, 5, 3, 5, 2, 6, 3\}, P = \{2, 5\}$$

$$t_5 = 52 \bmod 5 = 2,$$

$$t_6 = 26 \bmod 5 = 1,$$

$$t_7 = 63 \bmod 5 = 3$$

$$t_{i+1} = 10 * (t_i - T[i+1] * 10) + T[i+|P|+1] \% q$$

So only $s=1$ is match

Rabin-Karp algorithm

Run time? (Average? Worst case?)

Rabin-Karp algorithm

Run time?

- “preprocessing” (first loop) = $O(|P|)$
- “matching” (second loop) = $O(|T|)$

So $O(|T| + |P|)$ and as $n > m$, $O(|T|)$ on average

Worst case: always a match $O(|T| |P|)$