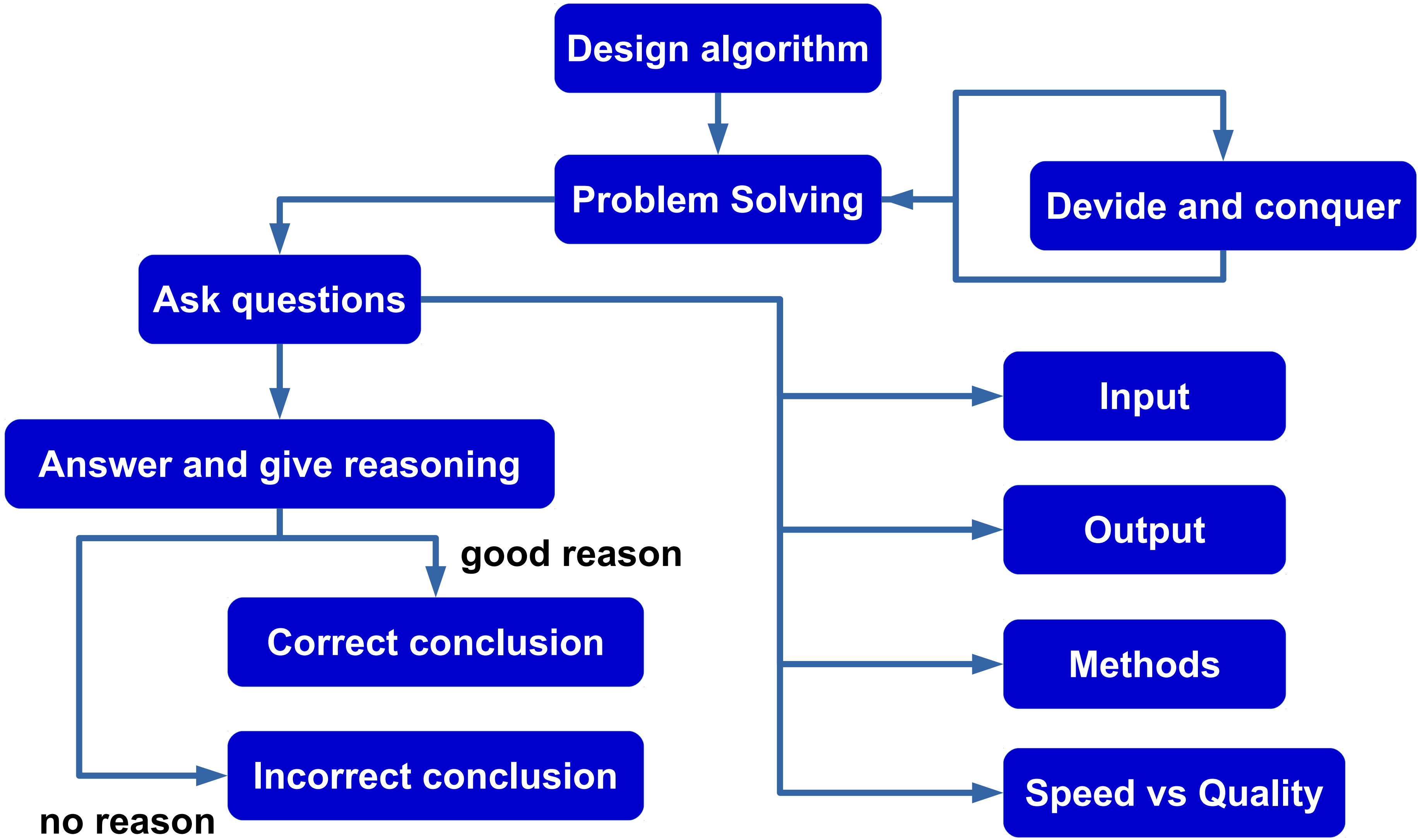
Advanced Programming Techniques

Benkedadra Mohamed
Benkorreche Mohammed El Amine
Youcefi Mohammed Yassine

Group 1 – M1 ISI
University of Mostaganem
Jan 2019

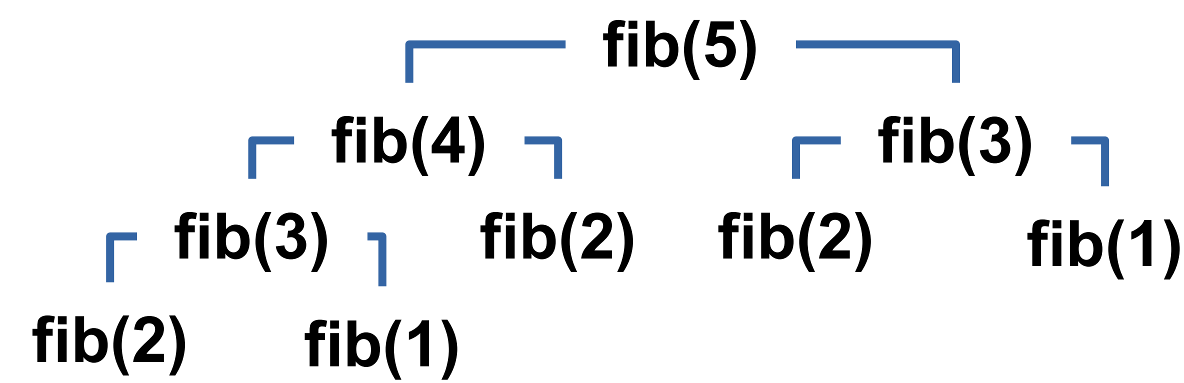
The Algorithm Design Manual



Introduction to Algorithms (Dynamic Programming)

fib(n-1) + fib(n-2)

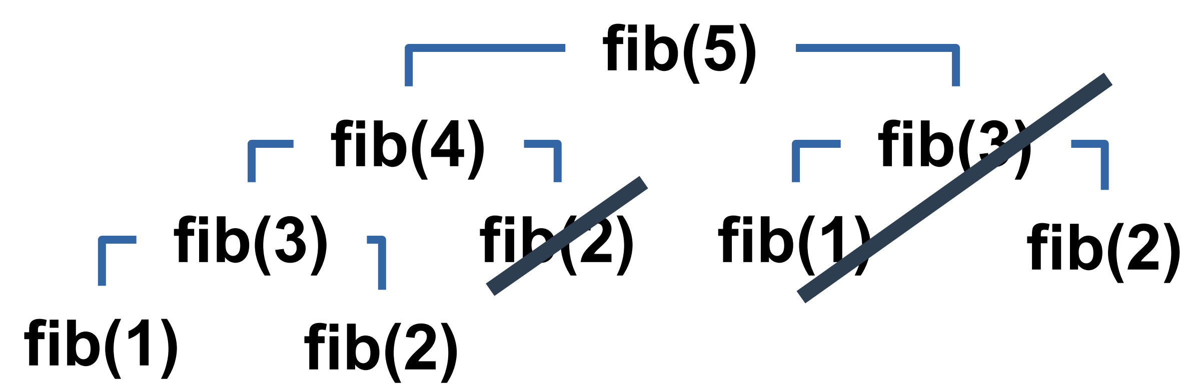
$O(2^n)$ experiential



If n-1 not in table :
    table[n-1] = fib(n-1)

If n-2 not in table :
    table[n-2] = fib(n-2)

table[n-1] + table [n-2]



O(n) linear

1	2	3	4	5

Single Threading

$$\begin{bmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a1*x+b1*y+c1*z \\ a2*x+b2*y+c2*z \\ a3*x+b3*y+c3*z \end{bmatrix}$$

Multi Threading

$$\begin{bmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Shared Memory

T1 – child

$$i = \begin{bmatrix} a1 & b1 & c1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

T2 – child

$$j = \begin{bmatrix} a2 & b2 & c2 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

T3 – child

$$k = \begin{bmatrix} a3 & b3 & c3 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Main Thread

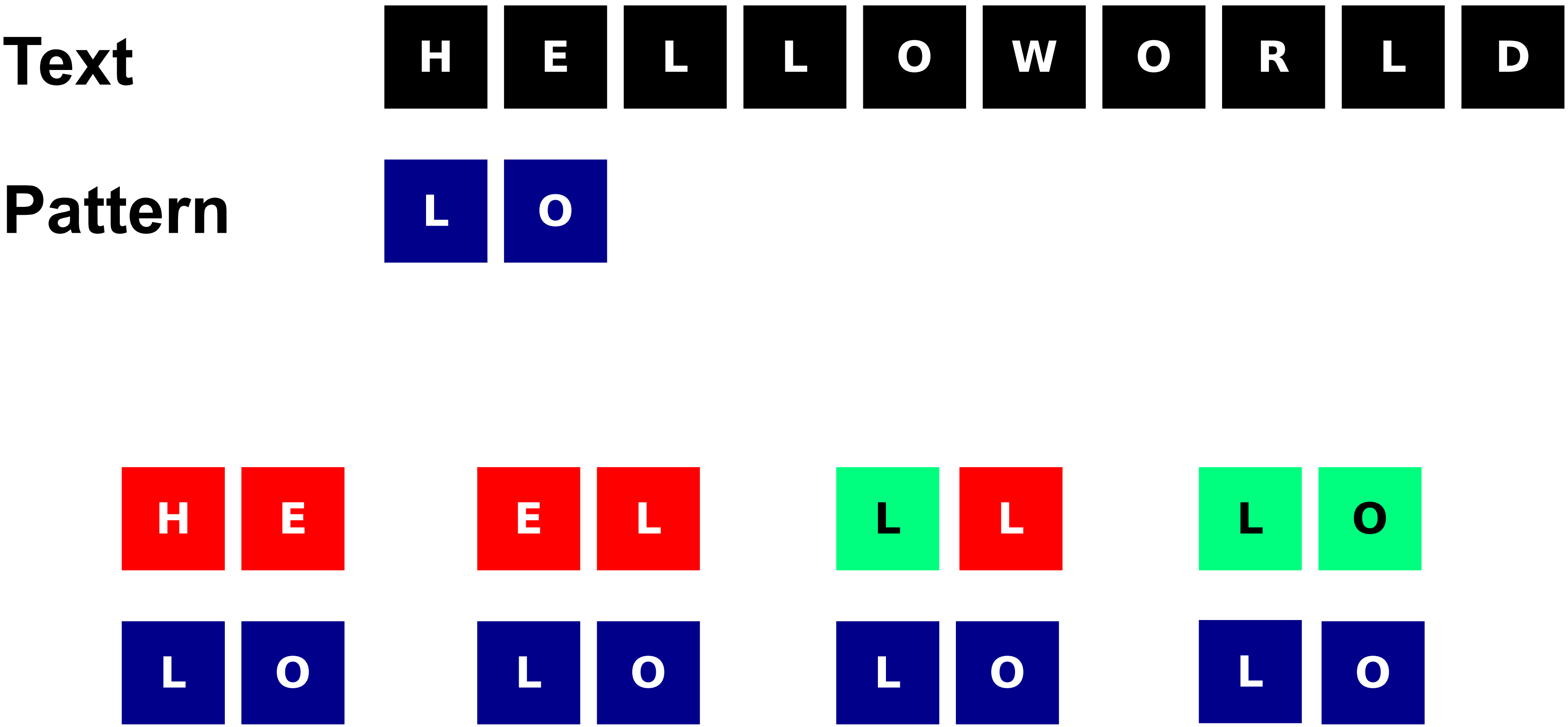
$$\begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

The Rabin Karp String Matching Algorithm

Rabin-Karp

- Also Known as Karp-Rabin algorithm
- Created by **Richard M. Karp** and **Michael O. Rabin**
- Uses **Hashing** to match patterns in text
- Where ***n*** is the length of the text to search in, and ***m*** is the length of the searched pattern:
    - Best case complexity is **O(n + m)**
    - Worst case complexity is **O(nm)**
- Often used to detect plagiarism

String Matching



Hashed String Matching

1	2	3		4	5		6		7
H	E	L		O	W		R		D

Text	H	E	L	L	O	W	O	R	L	D
Pattern	L	O								

$$H(s) = \sum_{i=0}^{n} char_i$$

L O

3 + 4 = 7

H E	1 + 2 = 3	=>	3 ≠ 7
E L	2 + 3 = 5	=>	5 ≠ 7
L L	3 + 3 = 6	=>	6 ≠ 7
L O	3 + 4 = 7	=>	7 = 7

L O

L O

Collision

1	2	3		4	5		6		7
H	E	L		O	W		R		D

Text	H	E	L	L	O	W	O	R	L	D
Pattern	H	O								

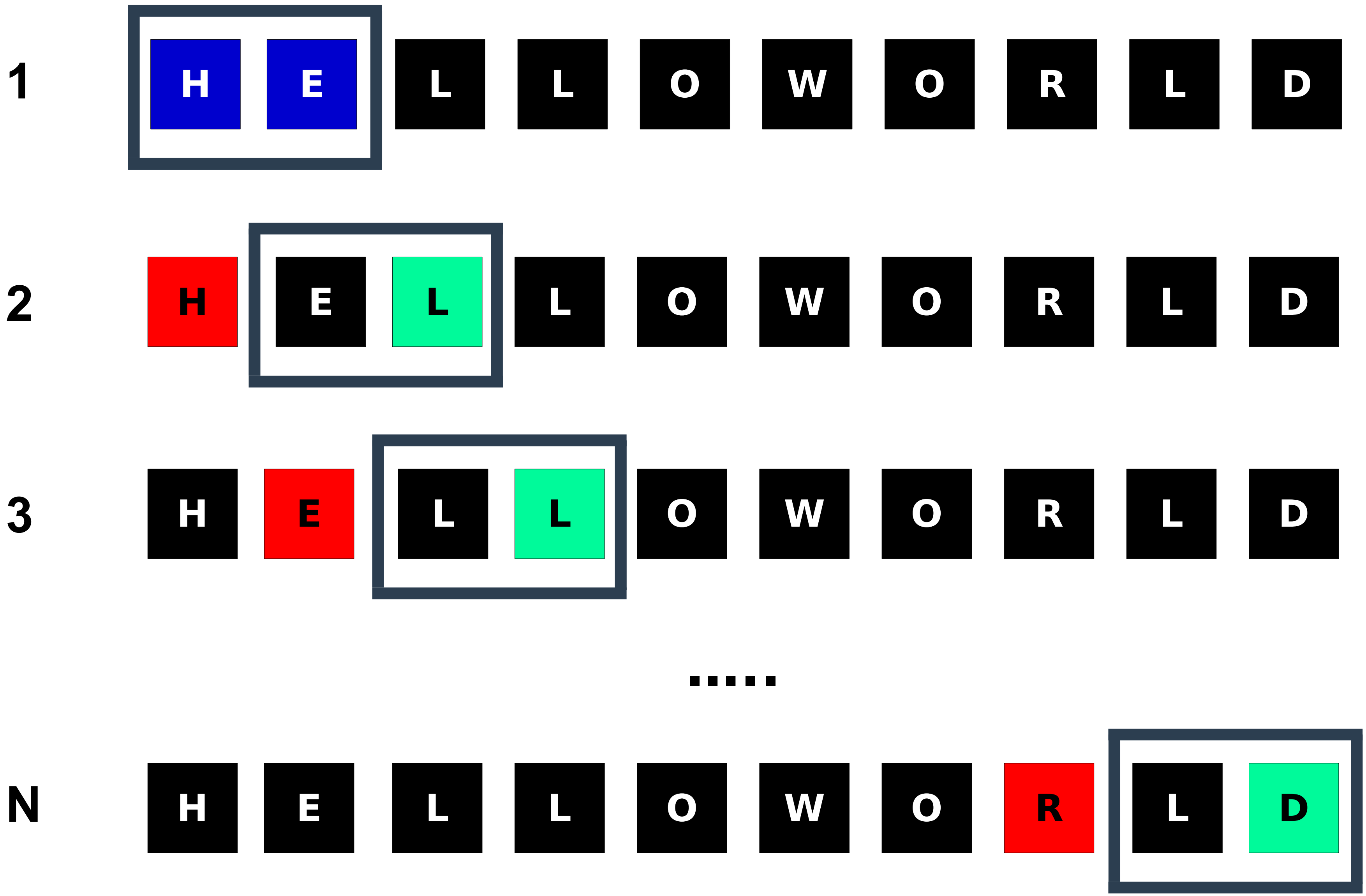
H	O	$1 + 4 = 5$
		$=$
E	L	$2 + 3 = 5$

Choosing a bad hashing function results in a high probability of a collision (finding equal hashes for different patterns)

Rolling Hash

- A function that allows calculating a new hash from an old one.
- Faster than a normal hash function.
- Usually has a polynomial or logarithmic complexity but varies from a function to another.
- Some rolling hash functions :
    - Polynomial rolling hash
    - Rabin fingerprint
- Can be used to slice content into pieces for easier processing.

Rolling Hash



Rabin Fingerprint

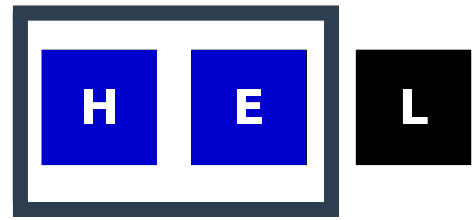
1 – define your alphabet's length

```
alpha_len = 2097152
```

2 – choose a random prime number

```
prime = 101
```

3 – calculate initial hash window




```
hash_val = 0
for char in pattern:
    hash_val = ( alpha_len * hash_val + ord(char) ) % prime
```

let's hash the string 'he' / in UTF-8 : h is 104, e is 101

hash('h') = (2097152 * 0 + 104) % 101 = 3

hash('he') = (2097152 * hash('h') + 101) % 101 = 65

Rabin Fingerprint

4 – calculate the H value

$$\mathit{alphabet\ length}^{\mathit{initial\ window\ length}-1}$$

```
h = pow(alpha_len, initial_window_length - 1)
```

Examples :

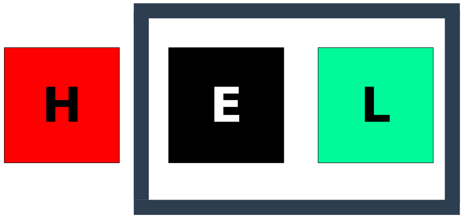
For a 3 characters window

Utf-8	$h = 2097152^{3-1}$	$= 2097152^{2}$	=	4398046511104
Ascii	$h = 256^{3-1}$	$= 256^{2}$	=	65536

Rabin Fingerprint

5 - Loop over the rest of the string and recalculate a new hash for each new window using the hash of the previous one

```
new_hash = old_hash - ( ord(old_char) * h )
new_hash = (new_hash  * alpha_len ) + ord(new_char)
new_hash = new_hash % prime
```



let’s hash the string ‘el’ in UTF-8 :

h is 104, e is 101, l is 108 / hash(‘he’) = 65 / h = 2097152

hash(‘el’) = [[65 - (104 * h)] * 2097152 + 108] % 101

hash(‘el’) = 68

Rabin Karp String Matching

1 – initialization

```
res = []
pl = len(pattern)
tl = len(text)
h = pow(alpha_len, pl - 1)
```

2 – hashing the searched pattern and the first window

```
# hash value of pattern
pattern_hash = calc_hash(pattern, prime)
# first window hash
win_hash = calc_hash(text[:pl], prime)
```

Rabin Karp String Matching

4 – Sliding the window

```
# windows sliding
for i in range(0, tl - pl - 1):
    if pattern_hash == win_hash:
        if pattern == text[i: i + pl]:
            res += [i]
            print('Pattern "{}" matched at {}'.format(pattern, i))

    ## next window hash val
    win_hash = recalc_hash(
        old_char = text[i],
        old_hash = win_hash,
        new_char = text[i + pl],
        h = h,
        prime = prime
    )
```

Rabin Karp String Matching

Examples

normal use

```
python3 main.py -p "word" -t "this is a word, and word sure word"
```

```
Pattern 'word' found at positions [10, 20, 30]
```

piping a text file

```
cat text.txt | python3 main.py -p "word" -t
```

Information

Resources

brilliant.org/wiki /rabin-karp-algorithm

Rabin Karp String Search Algorithm (Book)

Code

github.com/LogX7/rabin_karp

Contact

M.Benkedadra	hammicristo@gmail.com
Y.Youcefi	yanilacamora@gmail.com
M.A.Benkorreche	amine.benk27@gmail.com