The Comparison of Deterministic and Randomized String Matching Algorithm

By Ian Sun

Problem Definition

- String Matching != Approximate String Matching
- String Matching: Find if a pattern P exists in a text T.

  e.g. Find P = “random” in T = “I love randomized algorithm”

  Return true or 8(pos)
- Approximate String Matching: Find how similar two strings are.

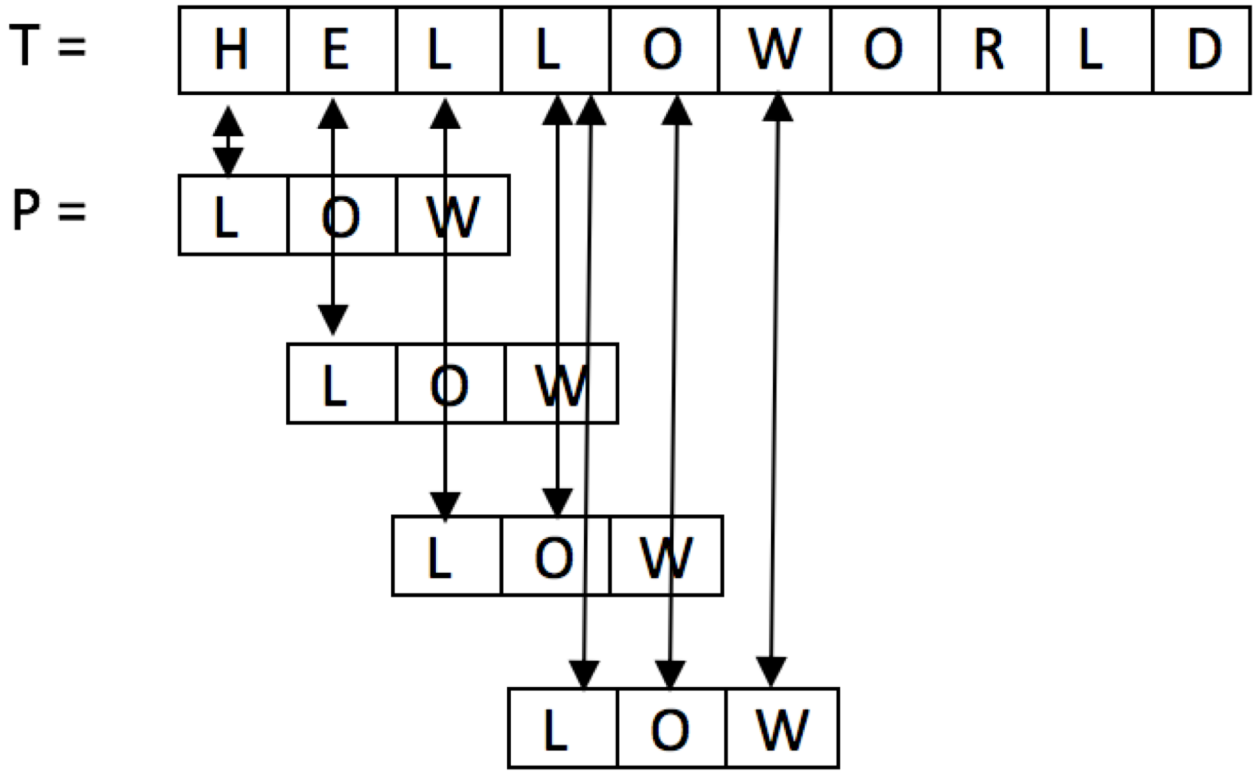
  e.g. T1 = “I love randomized algorithm”

  T2 = “I love algorithm”

  Return 16 or percentage

Naïve Algorithm (Deterministic)

- Naive algorithm basically compares pattern with subtext with the same length from left to right until it finds a match.



Rabin-Karp Algorithm(Randomized)

- Idea: Comparing with two strings ⟶ Comparing with two numbers.
- Hash a string into a number(Not efficient to calculate)

$$h(T_i) = (A(t_i)k^{m-1} + A(t_{i+1})k^{m-2} + \ldots + A(T_{m+i-1}k^0) mod p_n$$

- Rolling Hash(Efficient)

$$h(T_{i+1}) = (h(T_i) - k^m * \text{first digit of} T_i) * k + \text{next digit after } T_i$$

Example

T =	4	8	9	0	2	1	0	7

P =	9	0	2	1	0

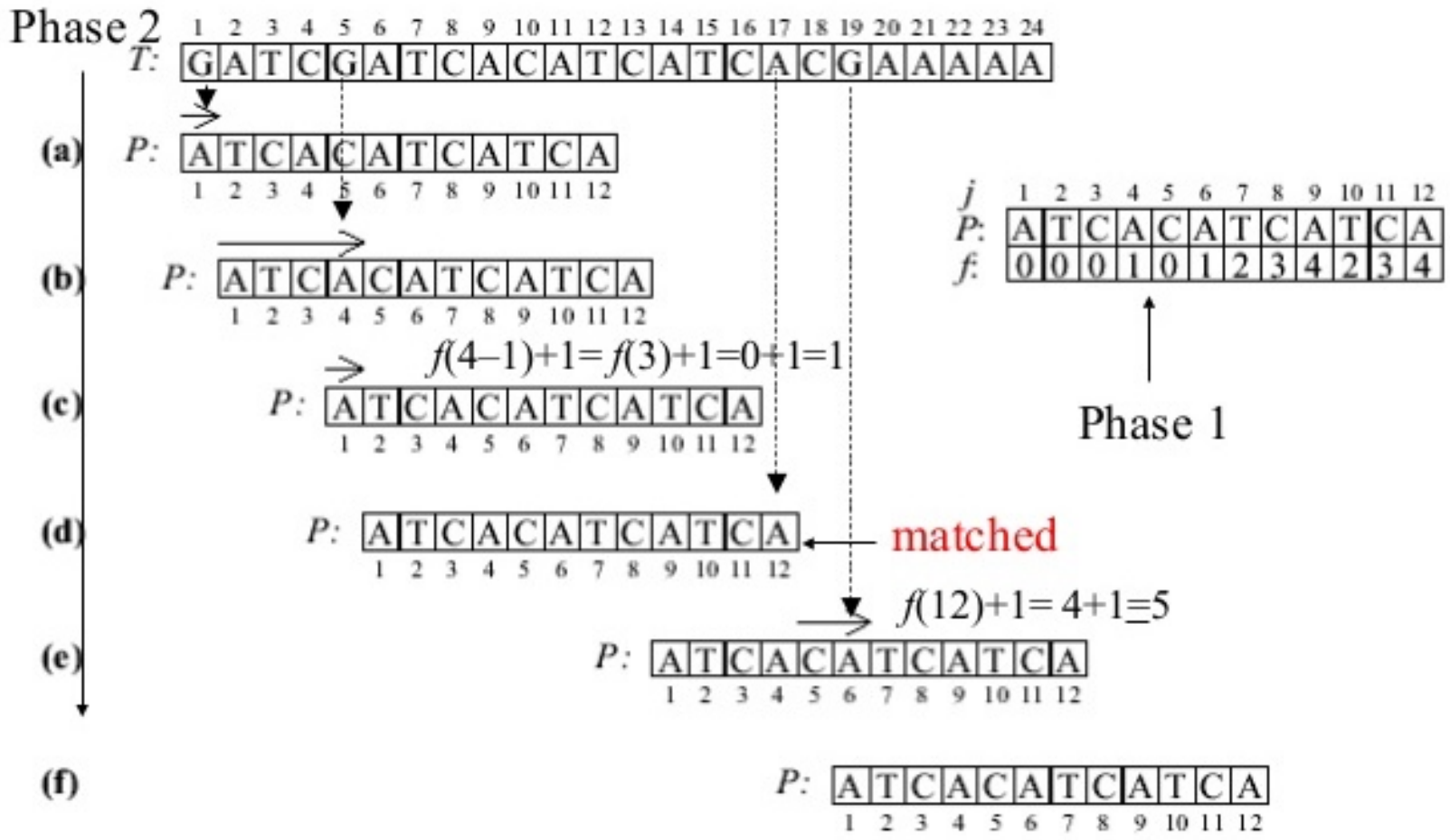
$T_0 = 4\ 8\ 9\ 0\ 2 \bmod p_n$

$T_1 = 8\ 9\ 0\ 2\ 1 \bmod p_n$

$\Longrightarrow T_1 = (T_0 - 4*10^4) *10 +1$

Rabin-Karp Algorithm (Randomized)

KMP Algorithm (Deterministic)



Comparison

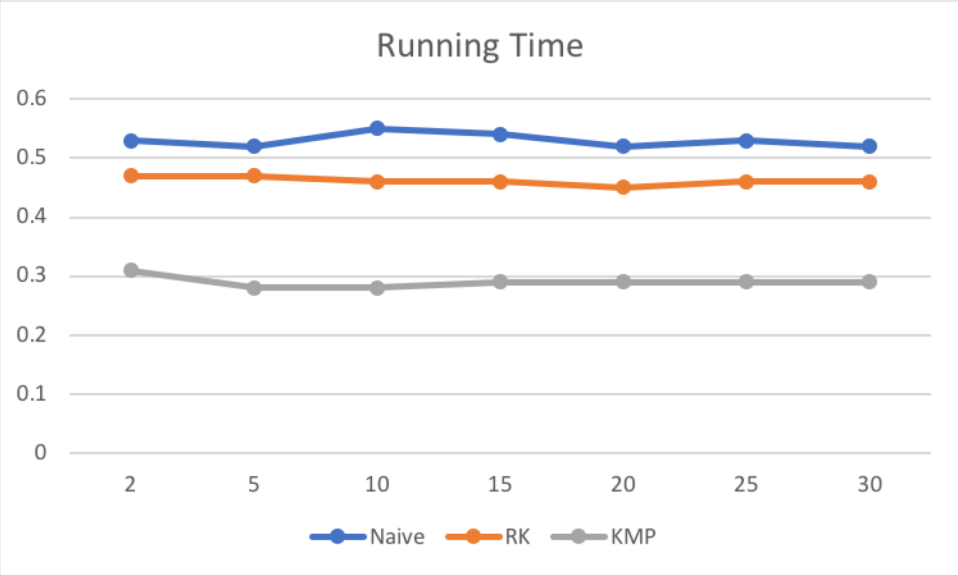
In theory, KMP performs the best overall and Naïve algorithm is the worst.

The time complexity of Rabin-Karp which wanders between O(m) and O(nm) basically depends on how many collisions in hash table.

Algorithm	Preprocessing	Running time
Naive	0	O(nm)
KMP	O(m)	O(n)
Rabin-Karp (rolling hash)	O(m)	O(n)~O(nm)

Experiment(text length fixed)

- With the help of Random Password Generator, I generate 100 texts, each with 10000 characters arbitrarily chosen from lowercase, uppercase and numbers. Then 100 patterns with length from 2 to 30 are generated with the same generator.
- X: length of pattern Y: Running Time



Algorithm	Preprocessing	Running time
Naive	0	O(nm)
KMP	O(m)	O(n)
Rabin-Karp (rolling hash)	O(m)	O(n)~O(nm)

This doesn't follow the table. Why?

Experiment(Only first pattern)

- In the program before, we want every pattern to be found, so if only finding the first matching pattern, we wonder if the result will be different.

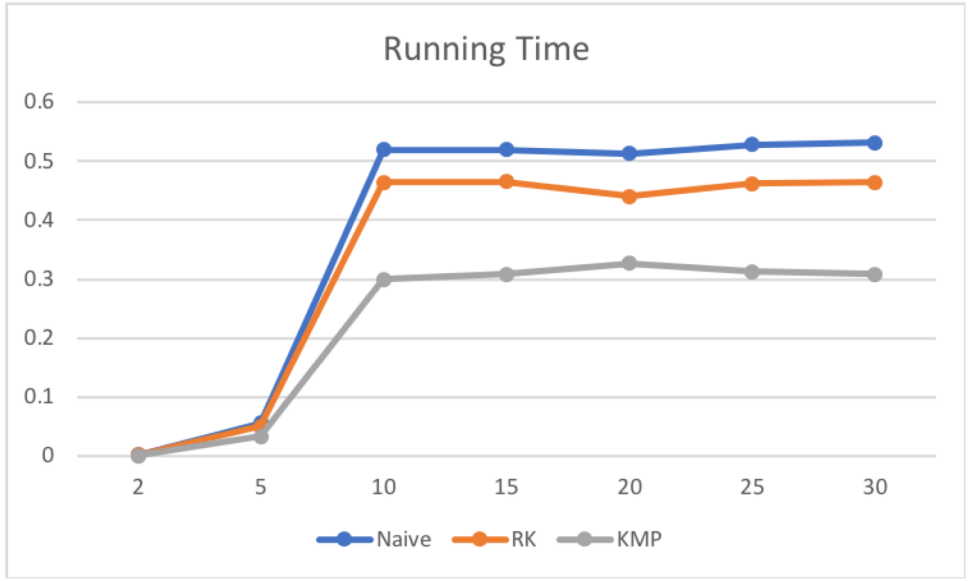


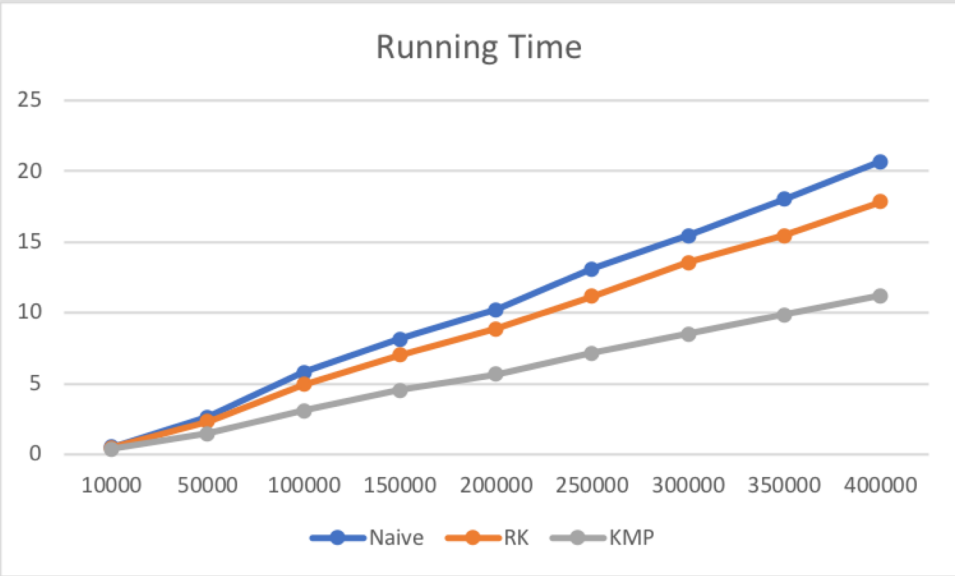
Table 1: Table of results in milliseconds - Alphanumeric Alphabet

$\|pattern\|$	matches	Naive	KMP	BM	RK
3	40	225	221	242	194
10	0	224	221	82	194
50	0	224	221	25	194

As the length of pattern increases, with greater probability that we can not find a match in the text.

Experiment(pattern length fixed)

- This time, we generate 100 patterns with 5 character and text length from 10,000 to 400,000.
- X: Length of text, Y: Running time



Algorithm	Preprocessing	Running time
Naive	0	O(nm)
KMP	O(m)	O(n)
Rabin-Karp (rolling hash)	O(m)	O(n)~O(nm)

Conclusion

- KMP is the best but hard to think of and implement
- Rabin-Karp is easy to think of and easy to implement
- Naïve is easy to think of and implement but not efficient

Randomized algorithm:

Easy to think of and reasonable time complexity.

One more thing

- Naïve algorithm implementation in python

```
def NaiveSearch1(P,T):
    result = []; m = len(T); n = len(P)
    for i in range(m-n+1):
        if T[i:i+n] == P:
            result.append(i)
```

Output graph:

Reason: Python compiler