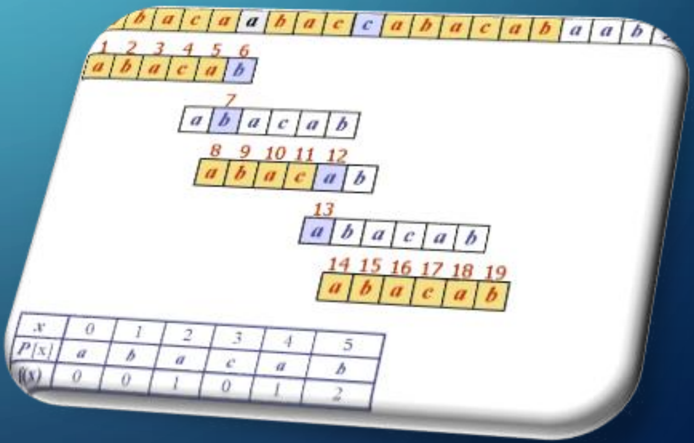
KNUTH-MORRIS-PRATT (KMP)

PATTERN MATCHING



x	0	1	2	3	4	5
P[x]	a	b	a	c	a	b
f(x)	0	0	1	0	1	2

Algorithm Complexity

- String matching efficiency is determined by the number of comparisons.

- Let *M* be the length of the pattern

- Let *N* be the length of the text

Brute Force Algorithm

- Search Text for first character in Pattern

- After finding first Character match, check the second Character in the Pattern to the Text.

- If a match, check next Character until end of Pattern or a mis-match occurs.

- BigO(M*N)

Prefix definition

- All the characters in a string with one or more cut off to the end.

- Example String: Ayyyyy
- Prefixes: A, Ay, Ayy, Ayyy, Ayyyy

Suffix Definition

- All the characters in a string, with one or more cut off the beginning.

- Example String: Lmaoo
- Suffixes: maoo, aoo, oo, o

Prefix Table Algorithm

The length of the longest PREFIX that matches a SUFFIX in the same sub-pattern.

```
Algorithm KMP-PrefixTable(P)
    m = |P|  // m is pattern length
    T[1] = 0  // T is prefix table
    i = 0  // i is longest prefix
    for j = 2 limit(m) step 1  // j is current index
        while i > 0 and P[i+1] != P[j]
            i = T[i]
        if P[i+1] = P[j] then
            i = i + 1
        T[j] = i
    return T
```

KMP Prefix Table

- DNA alphabet = { A, C, G, T }
- Pattern: **A C A C A G T**
- Text: ACAT ACGACACAGT

Prefix Table Construction

A

A C A C A G T

Prefixes 0

Suffixes 0

Prefix: 0

Suffix: 0

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Pattern	A	C	A	C	A	G	T
Prefix	**0**						

Prefix Table Construction

AC

ACACAGT

Prefixes 0

Suffixes 0

Prefix: A

Suffix: C

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**0**					

Prefix Table Construction

A C A

A C A C A G T

Prefixes 0

Suffixes 0

Prefix: A, AC

Suffix: A, CA

A is a prefix and suffix of length 1

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**0**	**1**				

Prefix Table Construction

A C A C

A C A C A G T

Prefixes 0

Suffixes 0

Prefix: A, AC, ACA

Suffix: C, AC, CAC

A is a prefix and suffix of length 2

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**1**	**1**	**2**			

Prefix Table Construction

A C A C A

A C A C A G T

Prefixes 0

Suffixes 0

Prefix: A, AC, ACA, ACAC

Suffix: A, CA, ACA, CACA

ACA is a prefix and suffix of length 3

Index	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**1**	**1**	**2**	**3**		

Prefix Table Construction

ACACAG

ACACAGT

Prefixes 0

Suffixes 0

Prefix: A, AC, ACA, ACAC, ACACA

Suffix: G, AG, CAG, ACAG, CACAG

No Match 0

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**0**	**1**	**2**	**3**	**0**	

Prefix Table Construction

ACACAGT

ACACAGT

Prefixes 0

Suffixes 0

Prefix: A, AC, ACA, ACACA, ACACAG

Suffix: T, GT, AGT, CAGT, ACAGT, CACAGT

No Match 0

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	0	0	1	2	3	0	0

KMP-Matcher Algorithm

```
Algorithm KMP-Matcher(S,P)
n = |S|  // n is search string length
m = |P|  // m is pattern length P
T = KMP-PrefixTable(P)
i = 0  // i is longest prefix found
for j = 1 limit(n) step 1
    while i>0 and P[i+1] != S[j]
        i = T[i]
    if P[i+1] = T[j]
        i = i + 1
    if i = m
        Output( S[j->m] )
        i = T[i]
```

Algorithm

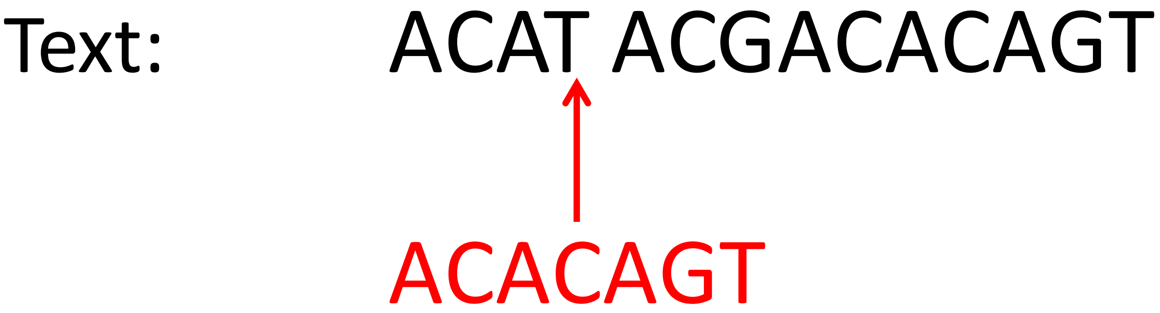
Pattern: ACACAGT

Text: ACAT ACGACACAGT

Prefix Table

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**0**	**1**	**2**	**3**	**0**	**0**

Algorithm



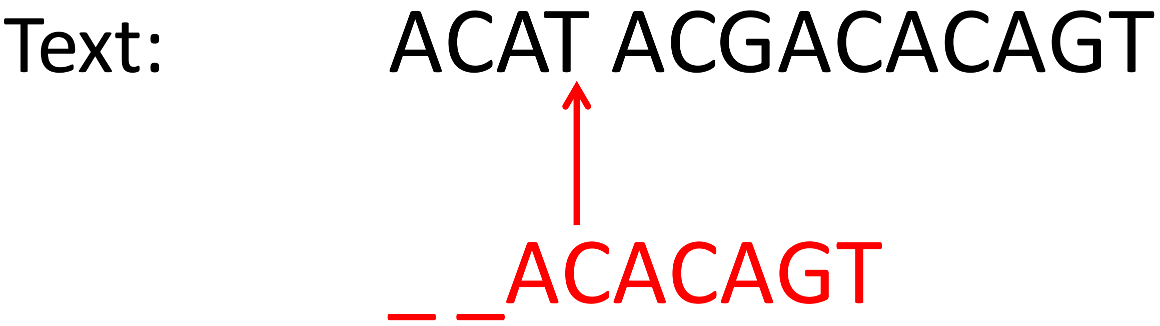
Text: ACAT ACGACACAGT

ACACAGT

Mismatch at C != T. Lookup C(4), shift 2

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	**0**	**0**	**1**	**2**	**3**	**0**	**0**

Algorithm



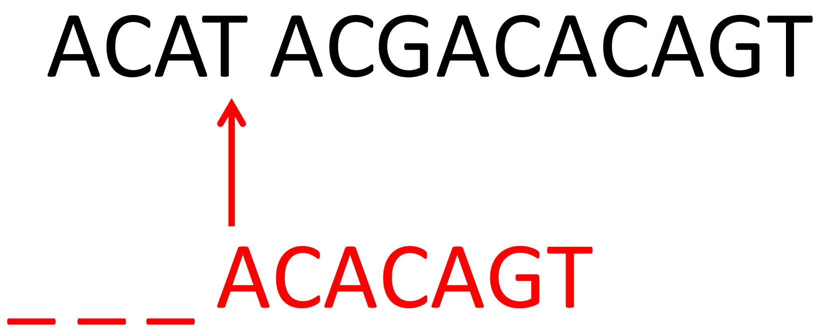
Text: ACAT ACGACACAGT

_ _ACACAGT

Mismatch at C != T. Lookup C(2), prefix 0, shift 1

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	0	0	1	2	3	0	0

Algorithm



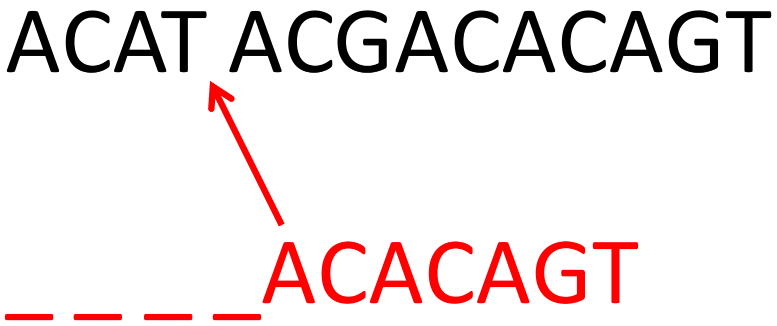
Text: ACAT ACGACACAGT

_ _ _ ACACAGT

Mismatch at A != T. Lookup A(1), prefix 0, shift 1

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	0	0	1	2	3	0	0

Algorithm



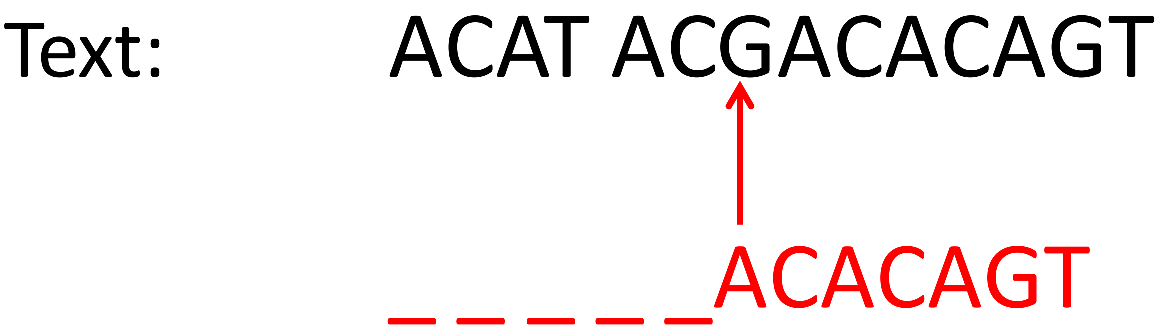
Text: ACAT ACGACACAGT

_ _ _ _ACACAGT

Mismatch at A != ‘ ‘. Lookup A(1), prefix 0, shift 1

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	0	0	1	2	3	0	0

Algorithm



Text: ACAT ACGACACAGT

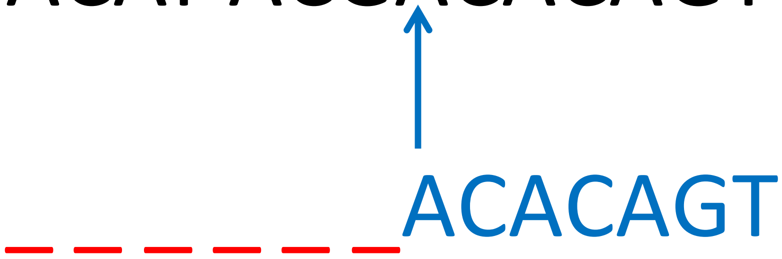
_ _ _ _ _ACACAGT

Mismatch at A != G. Lookup A(3), prefix 1, shift 1

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	0	0	1	2	3	0	0

Algorithm

Text: ACAT ACGACACAGT



ACACAGT

A == A → Matches at length.

Index	1	2	3	4	5	6	7
Pattern	A	C	A	C	A	G	T
Prefix	0	0	1	2	3	0	0

Time Complexity

- Worst case: All Prefix values 0 is O(N*M)

  N = length of text

  M = length of pattern

- Best case: O(N+M)

- Since usually N is significantly greater than M, O(N) is complexity.