**Question 1. Brute Force Algorithm**

THERE\_IS\_MORE\_TO\_LIFE\_THAN\_INCREASING\_ITS\_SPEED

GANDHI

m: Length of pattern

n: Length of text

for i 0 to n – m: Up to n – m + 1 iterations

Worst case scenario:

Best case:

Question 2. Horspool’s Algorithm

Horspool’s algorithm computes a Shift Table: The distance from the right most character to any other character.

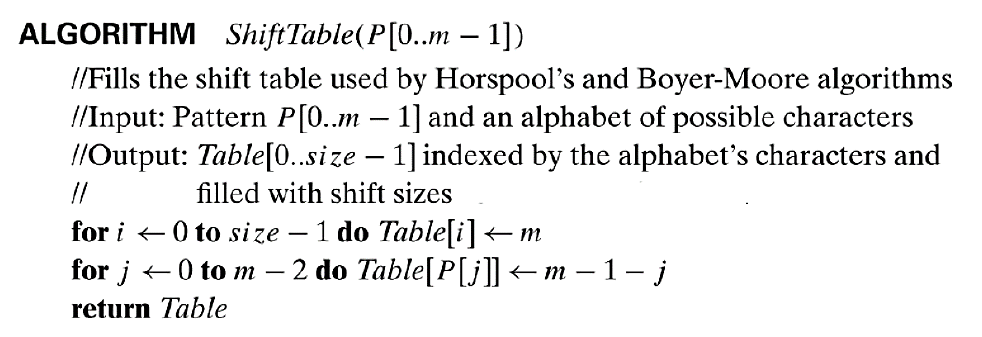
For example, given a pattern of length 10:

**TCCTATTCTT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Character | T | C | A | \* |
| Expected shifts | 1 | 2 | 5 | 10 |
| Actual shifts | 1 | 2 | 5 | 10 |

\* is any other character (doesn’t exist in our pattern)

Calculate the actual shift using the algorithm:



When j = 0, T becomes 10 – 1 – 0 = 9

When j = 1, C becomes 10 – 1 – 1 = 8

When j = 2, C becomes 10 – 1 – 2 = 7

When j = 3, T becomes 10 – 1 – 3 = 6

When j = 4, **A** becomes 10 – 1 – 4 = **5**

When j = 5, T becomes 10 – 1 – 5 = 4

When j = 6, T becomes 10 – 1 – 6 = 3

When j = 7, **C** becomes 10 – 1 – 7 = **2**

When j = 8, **T** becomes 10 – 1 – 8 = **1**

When j = 9, terminate, since j goes from 0 to 10 – 2 = 8.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Character | T | C | A | \* |
| Expected shifts | 1 | 2 | 5 | 10 |
| Actual shifts | 1 | 2 | 5 | 10 |

Apply on the following DNA Sequence:

m – 1 = 9

TTATAGAT**CT**CGTATTCTTTTATAGATCTCCTATTCTT

TCCTATTC**TT**

Mismatch. Shift by T = 1 character

TTATAGATCT**C**GTATTCTTTTATAGATCTCCTATTCTT

TCCTATTCT**T**

Mismatch. Shift by C = 2 characters

TTATAGATCTC**GT**ATTCTTTTATAGATCTCCTATTCTT

TCCTATTC**TT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGT**A**TTCTTTTATAGATCTCCTATTCTT

TCCTATTCT**T**

Mismatch. Shift by A = 5 characters

TTATAGATCTC**GTATTCTT**TTATAGATCTCCTATTCTT

TC**CTATTCTT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGTATTC**TTT**TATAGATCTCCTATTCTT

TCCTATT**CTT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGTATTCT**TTU**ATAGATCTCCTATTCTT

TCCTATT**CTT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGTATTCTTTT**A**TAGATCTCCTATTCTT

TCCTATTCT**T**

Mismatch. Shift by A = 5 characters

TTATAGATCTCGTATTCTTTTATAG**AT**CTCCTATTCTT

TCCTATTC**TT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGTATTCTTTTATAGAT**C**TCCTATTCTT

TCCTATTCT**T**

Mismatch. Shift by C = 2 characters

TTATAGATCTCGTATTCTTTTATAGAT**CT**CCTATTCTT

TCCTATTC**TT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGTATTCTTTTATAGATCT**C**CTATTCTT

TCCTATTCT**T**

Mismatch. Shift by C = 2 characters

TTATAGATCTCGTATTCTTTTATAGATCTC**CT**ATTCTT

TCCTATTC**TT**

Mismatch. Shift by T = 1 character

TTATAGATCTCGTATTCTTTTATAGATCTCCT**A**TTCTT

TCCTATTCT**T**

Mismatch. Shift by A = 5 characters

TTATAGATCTCGTATTCTTTTATAGATC**TCCTATTCTT**

**TCCTATTCTT**

Pattern found at index 28.

Question 3: Boyer-Moore’s Algorithm

Pattern

**00001**

Compute two tables: Bad Symbol, Good Suffix

Bad Symbol Table (the same as Horspool’ table)

|  |  |  |  |
| --- | --- | --- | --- |
| Character | 0 | 1 | \* |
| Shift | 1 | 5 | 5 |

Good Suffix Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 01 | 001 | 0001 |
| Shift | 5 | 5 | 5 | 5 |

For 1000 zeros, always shift by 1 (since the bad symbol is always 0).

Number of comparisons: n – m + 1= 1000 – 5 + 1 = 996 comparisons (worst case).

Pattern

**10000**

Compute two tables: Bad Symbol, Good Suffix

Bad Symbol Table (the same as Horspool’ table)

|  |  |  |  |
| --- | --- | --- | --- |
| Character | 0 | 1 | \* |
| Shift | 1 | 4 | 5 |

Good Suffix Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 00 | 000 | 0000 |
| Shift | 3 | 2 | 1 | 5 |

**00000000000000…00000000**

**10000**

Good suffix, shift by 5

Good suffix, shift by 5

…

Always get a good suffix that is of the longest size.

So, we will not get a mismatch until all characters in the pattern has been compared with all characters in the string.

Even with fewer shifts, we are making MORE comparisons -> Every character in the text is compared n = 1000 comparisons.