<script src="./themes/cycle.js"> </script>

# Tutorial 11: String Matching Algorithms

**CAB301 - Algorithms and Complexity**

School of Computer Science, Faculty of Science

## Agenda

1. **Lecture Recap**: String Matching Algorithms
    - Brute Force Algorithm
    - Horspool's Algorithm
    - Boyer-Moore's Algorithm
2. **Tutorial Questions + Q&A**

## String Matching Algorithms

Given a text $T$ and a pattern $P$, find all occurrences of $P$ in $T$. This normally involves finding the starting index of the first occurrence of $P$ in $T$.

Example:

- $T =$ `Goodbye, CAB301!`
- $P =$ `CAB301`

| **Character** | **G** | **o** | **o** | **d** | **b** | **y** | **e** | **,** | | **C** | **A** | **B** | **3** | **0** | **1** | **!** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Here, `CAB301` starts at index 9.

## Brute Force Algorithm

**Idea**: Compare each character of the pattern $P$ with the text $T$.

Example:

Find `NET` in `INTERNET`.

`INTERNET`

<div class='cycle'>

`NET`: $(i = 0)$ - $\texttt{I} \neq \texttt{N}$, so move the pattern

`  NET`: $(i = 1)$ - $\texttt{N} = \texttt{N}$, so check next character, $\texttt{T} \neq \texttt{E}$, so move the pattern

`    NET`: $(i = 2)$ - $\texttt{T} \neq \texttt{N}$, so move the pattern

# Horspool's Algorithm - Shift Table

Instead of shifting the pattern by one character on mismatch, **shift strategically**.

From the pattern, precompute a **shift table**, which determines how far to shift the pattern on mismatch.

For each character in the pattern, the value in the shift table is the **distance from the rightmost occurrence** of that character **to the last character** of the pattern (**except for the last character**). Otherwise, the shift is the **length of the patter**n.

**Example**: `barbaric`. Any other character not in the pattern denoted by `*`.

| **Character** | **a** | **b** | **c** | **i** | **r** | ***** |
|---|---|---|---|---|---|---|
| **Shift** | 3 | 4 | 8 | 1 | 2 | 8 |

# Horspool's Algorithm - Execution

Compare the text with the pattern from **right to left**. If mismatch, look up the last character of the mismatched substring in the shift table to determine the shift.

**Example**: Find `barbaric` in `the_artic_sarcastic_barbaric_bar`.

<div class="flexbox"> <div>

```
the_artic_sarcastic_barbaric_bar
```

<div class='cycle'> <div>

```
       ↑
```

```
barbaric
```

Mismatch last character: `i`, so shift by 1.

## Boyer-Moore's Algorithm - Good Suffix Shift Table

**Idea**: Use the previous **shift table** (now called **bad symbol shift table**) and a **good suffix shift table** to determine the shift.

The **good suffix shift table** finds the distance to shift a suffix **to the next occurrence of the same suffix** in the pattern (if the character before the suffix is different).

If no occurrence of the full suffix is found, shift until the **tail of the suffix** matches.

Example: `taattaat`

| **Suffix** | **t** | **at** | **aat** | **taat** | **ttaat** | **attaat** | **aattaat** |
|---|---|---|---|---|---|---|---|
| **Shift** | 3 | 7 | 7 | 4 | 4 | 4 | 4 |

## Boyer-Moore's Algorithm - Execution

Similar to Horspool's Algorithm, but on mismatch, if a good suffix is found, shift using the **good suffix shift table**. Otherwise, shift using the **bad symbol shift table**.

**Example**: Find `taattaat` in `tgaccсttctatgggcgctccgatacgccgacttatccga`.

<div class="flexbox"> <div style="font-size: 24px">

```
tgacccttctatgggcgctccgatacgccgacttatccga
```

<div class='cycle'> <div>

```
      ↑↑
```

```
taattaat
```

Good suffix `t`, so shift by 3.