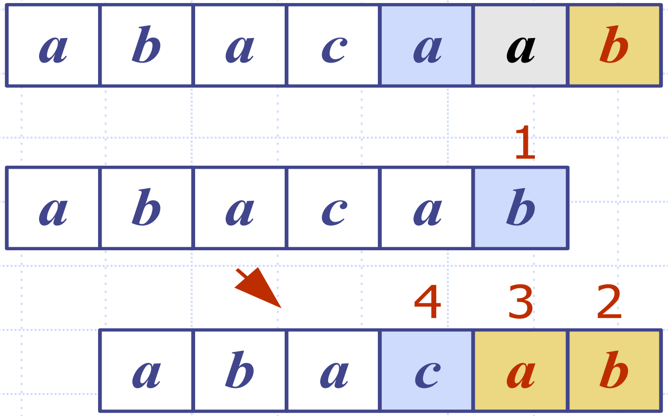
Pattern Matching



Strings

- A string is a sequence of characters
- Examples of strings:
  - Python program
  - HTML document
  - DNA sequence
  - Digitized image
- An alphabet Σ is the set of possible characters for a family of strings
- Example of alphabets:
  - ASCII
  - Unicode
  - {0, 1}
  - {A, C, G, T}

- Let P be a string of size m
  - A substring $P[i \,..\, j]$ of P is the subsequence of P consisting of the characters with ranks between i and j
  - A prefix of P is a substring of the type $P[0 \,..\, i]$
  - A suffix of P is a substring of the type $P[i \,..m - 1]$
- Given strings T (text) and P (pattern), the pattern matching problem consists of finding a substring of T equal to P
- Applications:
  - Text editors
  - Search engines
  - Biological research

Brute-Force Pattern Matching

- The brute-force pattern matching algorithm compares the pattern P with the text T for each possible shift of P relative to T, until either
  - a match is found, or
  - all placements of the pattern have been tried
- Brute-force pattern matching runs in time $O(nm)$
- Example of worst case:
  - $T = aaa \ldots ah$
  - $P = aaah$
  - may occur in images and DNA sequences
  - unlikely in English text

```
Algorithm BruteForceMatch(T, P)
    Input text $T$ of size $n$ and pattern
        $P$ of size $m$
    Output starting index of a
        substring of $T$ equal to $P$ or $-1$
        if no such substring exists
    for $i \leftarrow 0$ to $n - m$
        { test shift $i$ of the pattern }
        $j \leftarrow 0$
        while $j < m \wedge T[i + j] = P[j]$
            $j \leftarrow j + 1$
        if $j = m$
            return $i$ {match at $i$}
        else
            break while loop {mismatch}
    return -1 {no match anywhere}
```

Right to Left Matching

- Matching the pattern from right to left
- For a pattern abc:

T: `bbacdcbaabcddcdaddaaabcbcb`

P: `abc`

- Worst case is still O(n m)

Bad Character Rule (BCR) (1)

- On a mismatch between the pattern and the text, we can shift the pattern by more than one place.

```
ddbbacdcbaabcddcdaddaaabcbcb
acabc
```

Bad Character Rule (BCR) (2)

- Preprocessing –
  - A table, for each position in the pattern and a character, last occurrence of the mismatched character in P preceding the mismatch . O(n |Σ|) space. O(1) access time.

a	c	a	b	c
1	2	3	4	5

	1	2	3	4	5
a	1	1	3	3	3
b				4	4
c		2	2	2	5