COMP20007 Design of Algorithms

Input Enhancement Part 2: String Searching

Daniel Beck

Lecture 18

Semester 1, 2020

String Search - Recap

String Search - Recap

- Goal: given text of size n, find a string (pattern) of size m.

String Search - Recap

- Goal: given text of size n, find a string (pattern) of size m.
- Brute force algorithm: $O(m \times n)$.

String Search - Brute Force

not even in my pattern

Text

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

B A R B E R

pattern

→ $\boxed{\text{B A R B E R}}$ →

→ B A R B E R

→ ...

$\boxed{\text{B A R B E R}}$ →

Input Enhancement

- Longer shifts can be made if we know statistics about the pattern.

Input Enhancement

- Longer shifts can be made if we know statistics about the pattern.
- Same way we could sort arrays more efficiently by knowing statistics about the array (Counting Sort).

Input Enhancement

- Longer shifts can be made if we know statistics about the pattern.
- Same way we could sort arrays more efficiently by knowing statistics about the array (Counting Sort).
- The Horspool's algorithm use this idea to make string search faster.

Input Enhancement

- Longer shifts can be made if we know statistics about the pattern.
- Same way we could sort arrays more efficiently by knowing statistics about the array (Counting Sort).
- The Horspool's algorithm use this idea to make string search faster.
- Key idea: scan the text from left to right but scan the pattern from right to left.

Longer shifts - Case 1

The last character is not in the pattern.

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

B A R B E R

BARBER

Shift the whole pattern.

Longer shifts - Case 2

W - 6 positions
A - 4 positions

The last character does not match but it is in the pattern.

exist in the patter

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P

align the last A

B A R B E R

BARBER

Shift the pattern until the last occurence of the character.

Longer shifts - Case 3

The last character matches but one of the $m - 1$ characters does not match and the last character is unique.

it can't be anywhere else in the pattern

```
J I M _ S A [W] _ M E _ I N _ A _ B A R B E R S H O P
  S E E S A W
              S E E S A W
```

Shift the whole pattern.

Longer shifts - Case 4

The last character matches but one of the $m-1$ characters does not match and the last character is not uniques

align with the R before the last R

```
J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
                        R E O R D E R
```

R E O R D E R

Shift the pattern until the last occurence of the character.

Horspool - Preprocessing

- The number of allowed shifts depends on the character type only.

Horspool - Preprocessing

- The number of allowed shifts depends on the character type only.
- Horspool builds a dictionary with all characters in the alphabet and their corresponding allowed skips for a pattern.

Horspool - Preprocessing

- The number of allowed shifts depends on the character type only.
- Horspool builds a dictionary with all characters in the alphabet and their corresponding allowed skips for a pattern.

BARBER

5 4 3 2 1

character c	A	B	C	D	E	F	...	R	...	Z	_
shift $t(c)$	4	2	6	6	1	6	6	$\boxed{3}$	6	6	6

there has another R

Horspool - FindShifts

alphabet size $26+1=27$

function $\text{FindShifts}(P[0..m-1])$

  for $i \leftarrow 0$ to a **do** → initialise the dictionary

    $Shift[i] \leftarrow m$

  for $j \leftarrow 0$ to $m-2$ **do**

    $Shift[P[j]] \leftarrow m-(j+1)$

SEESAW

if no match shift 6

stop $\boxed{m-2}$ $m-1$

BARBER

$6-(0+1)$

SHIFT

A	~~6~~ 4
B	~~6~~ ~~5~~ 2
C	6
D	6
E	~~6~~ 1
⋮	⋮
R	~~6~~ 3
⋮	
Z	6
'_'	6

Horspool - Algorithm

function HORSPOOL($P[0..m-1]$, $T[0..n-1]$)

    FINDSHIFTS(P) → get the shift table

    $i \leftarrow m - 1$ → look at last charater

    while $i < n$ **do**

        $k \leftarrow 0$ end of patter mismatch

        while $k < m$ **and** $P[m-1-k] = T[i-k]$ **do**

            $k \leftarrow k + 1$

        if $k = m$ **then** ▷ We have a match

            return $i - m + 1$ ▷ Start of the match

        else

            $i \leftarrow i + Shift[T[i]]$ ▷ Slide the pattern along

    return -1

Horspool - Properties

- Worst-case still $O(m \times n)$.
- For random strings, it's linear and faster in practice compared to the brute force version.

Other String Search algorithms

- Boyer-Moore: extends Horspool to allow shifts based on suffixes.

Other String Search algorithms

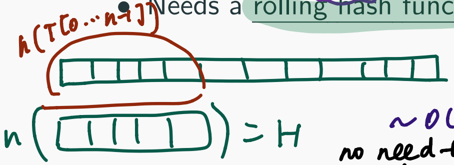
- Boyer-Moore: extends Horspool to allow shifts based on suffixes.
- Knuth-Morris-Pratt: also preprocess the patten but builds a finite-state automaton.

Other String Search algorithms

- Boyer-Moore: extends Horspool to allow shifts based on suffixes.
- Knuth-Morris-Pratt: also preprocess the patten but builds a finite-state automaton.
- Rabin-Karp: uses hash functions to filter negative matches.
  - Needs a rolling hash function to be efficient.

$h(T[i \cdots j]) = f(h(T[i-1, \cdots, j-1]))$

$f(j)$



$\sim O(n)$

no need to scan unless a match!

$h \neq H$

$h = H$ { collision / true match

good hash function $h \neq H$ most case

Summary and Practical Uses

- String search algorithms can be sped up by input enhancement.

Summary and Practical Uses

- String search algorithms can be sped up by input enhancement.
- Allocates extra memory to preprocess the pattern.

Summary and Practical Uses

- String search algorithms can be sped up by input enhancement.
- Allocates extra memory to preprocess the pattern.
- Horspool uses a dictionary of shifts.

Summary and Practical Uses

- String search algorithms can be sped up by input enhancement.
- Allocates extra memory to preprocess the pattern.
- Horspool uses a dictionary of shifts.
- The Boyer-Moore extension is one of the most used string searching algorithms, for instance in the “grep” ~~Linx~~ Linux command line tool.

Summary and Practical Uses

- String search algorithms can be sped up by input enhancement.
- Allocates extra memory to preprocess the pattern.
- Horspool uses a dictionary of shifts.
- The Boyer-Moore extension is one of the most used string searching algorithms, for instance in the “grep” Linx command line tool.

Next week: trade memory for speed by storing intermediate solutions.