CS 380
ALGORITHM DESIGN
AND ANALYSIS

Lecture 18: String Matching and Searching
Brute Force, Horspool

References:

String Matching / Searching

- Naive
- Horspool\(^1\)
- Boyer-Moore\(^1\)
- Rabin-Karp\(^2\) (Hash)
- Knuth Morris Pratt\(^2\) (DFA!)

\(^1\) Levitin, Introduction to The Design and Analysis of Algorithms, 3rd edition, Pearson Addison Wesley, p 259

\(^2\) CLRS, p 990 & 1002

Not Dynamic Programming because there are not subproblems.

But precompute a table to help you solve the problem.
Naïve: Example

• text: Asymptotic  pattern: tic

http://whocouldthat.be/visualizing-string-matching/

Naïve Search

**ALGORITHM**  
**BruteForceStringMatch**(*T*[0..*n* − 1], *P*[0..*m* − 1])  
//Implements brute-force string matching  
//Input: An array *T*[0..*n* − 1] of *n* characters representing a text and  
//an array *P*[0..*m* − 1] of *m* characters representing a pattern  
//Output: The index of the first character in the text that starts a  
//matching substring or −1 if the search is unsuccessful  
**for** *i* ← 0 **to** *n* − *m*  
**do**  
  *j* ← 0  
  **while** *j* < *m* and *P*[*j*] = *T*[*i* + *j*]  
  **do**  
    *j* ← *j* + 1  
  **if** *j* = *m* **return** *i*  
**return** −1
Naïve Search: Analysis

Suppose text.length = n, pattern.length = m

• Worse case?
  
  • \(O(m \times (n-m+1)) = O(mn)\) comparisons
  
• Best case:
  
  • \(\Omega(n)\) comparisons

Horspool

• Match the pattern right (last character) to left (first character)

• On mismatch, shift the pattern as far as you can
  
  • By 1+ character(s)

• Preprocess string to determine shifting
  
  • build a table for shifts for each valid character
Four Possibilities: Case 1

Assume on a particular character c in text
1) If c is not in the pattern, can shift pattern by its entire length to the right!

Four Possibilities: Case 2

2) If c is in the pattern but not in last position, shift pattern so character aligns with right-most occurrence of c in pattern
**Four Possibilities: Case 3**

3) If character c is **in the last position** and there are **no other** c’s in rest of pattern, shift pattern by its entire length to right

```
<table>
<thead>
<tr>
<th>text</th>
<th>character comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>drive</td>
</tr>
<tr>
<td></td>
<td>grove</td>
</tr>
</tbody>
</table>
```

**Four Possibilities: Case 4**

4) If c is **in the last position** but there are other c’s in pattern, align with the right most occurrence of c in first m-1 characters

```
<table>
<thead>
<tr>
<th>text</th>
<th>character comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pacific</td>
</tr>
<tr>
<td></td>
<td>pacific</td>
</tr>
</tbody>
</table>
```

```
Moral from Cases

- Always align character with its right most occurrence in first m-1 characters of pattern. If there is no such occurrence, shift pattern by its entire length to the right.

Shifting

\[ t(c) = \begin{cases} 
\text{the pattern's length } m, & \text{if } c \text{ is not among the first } m - 1 \text{ characters of the pattern;} \\
\text{the distance from the rightmost } c \text{ among the first } m - 1 \text{ characters of the pattern to its last character, otherwise.} 
\end{cases} \]  

(7.1)
**ShiftTable**

**Algorithm**  
`ShiftTable(P[0..m-1])`  
// Fills the shift table used by Horspool's and Boyer-Moore algorithms  
// Input: Pattern `P[0..m-1]` and an alphabet of possible characters  
// Output: `Table[0..size-1]` indexed by the alphabet's characters and filled with shift sizes computed by formula (7.1)  
```
for i ← 0 to size - 1 do Table[i] ← m
for j ← 0 to m - 2 do Table[P[j]] ← m - j
return Table
```

<table>
<thead>
<tr>
<th>P</th>
<th>a</th>
<th>c</th>
<th>i</th>
<th>f</th>
<th>i</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>...</td>
<td>f</td>
<td>...</td>
<td>i</td>
</tr>
</tbody>
</table>

After being a 3!

Pattern (m=7)

<table>
<thead>
<tr>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

---

**Horspool Algorithm**

**Algorithm**  
`HorspoolMatching(P[0..m-1], T[0..n-1])`  
// Implements Horspool's algorithm for string matching  
// Input: Pattern `P[0..m-1]` and text `T[0..n-1]`  
// Output: The index of the left end of the first matching substring or -1 if there are no matches  
// `ShiftTable(P[0..m-1])`  
// `Table[0..size-1]`  
```
i ← m - 1  
while i ≤ n - 1 do  
    k ← 0  
    while k ≤ m - 1 and P[m - 1 - k] = T[i - k] do  
        k ← k + 1  
    if k = m  
        return i - m + 1  
    else  
        i ← i + Table[T[i]]  
return -1
```
Horspool: Summary (p. 261)

1. Given text and pattern, construct the shift table
2. Align left side of pattern with left side of beginning of text.
3. Start with last character in pattern, start matching characters until all characters in pattern are matched OR you encounter a mismatch
4. If c is text character aligned with last character of pattern, shift pattern by t(c)
5. Repeat steps 3+4 until match or end of text.

Horspool analysis

m=pattern length, n=text length, k=alphabet length

- Preprocessing time: \( \theta(m+k) \)
- Searching time: Best: \( \Omega(n/m) \)
  Worst: \( O(nm) \)
  Average: \( \theta(n) \)

Trades space (construct ShiftTable) for time
Example

- String
  - GTACTAGAGGACGTATGTA
- Pattern:
  - ATGTA

- Generate the shift table
- Show the steps of the algorithm