Project 3: Pattern Matching Algorithms

ITCS 6114 – Algorithms and Data Structures

By Ria Banerjee & TJ Bah

# Implementation

1. ReadMe
2. Brute-Force (Naïve) Algorithm - project\_3\_Naive\_Algo.zip
3. Boyer-Moore-Horspool Algorithm - project\_3\_BM\_Horspool\_Algo.zip
4. Knuth-Morris-Pratt Algorithm – https://colab.research.google.com/drive/1\_d04gjSJLokszAEj2NODVgpzjVikZZg7

# Experiment

### Brute-Force (Naïve) Algorithm

1. **Description:**
   1. The naïve or Brute-Force pattern matching algorithm will compare an input pattern P with an input text T for every possible shift of P relative to T. This results in either a found match or all placements of the pattern have been attempted.
2. **Data Structures used:**
   1. String
   2. Array
3. **Runtime Analysis:**
   1. Pseudocode

**0** – Algorithm BruteForceMatch(T, P) ---------- --> O((n-m + 1) X m) = O(nm)

**1** - // Input text T of size n and pattern P of size m

**2** - // Output starting index of a substring of T equal to P or -1 if no such substring exists

**3** - for i = 0 to n – m ---------- --> O(n-m + 1)

**4** - { test shift i of the pattern }

**5** - j = 0

**6** - while j < m and 𝑻[𝒊 + 𝒋] = P[j] ---------- --> O(m)

**7** - j = j + 1

**8** - if j = m return i {match at i}

**9** - return -1 {no match anywhere}

1. **Sample Input text and patterns:**

// TEST Case 1

String T = "THIS IS A SIMPLE EXAMPLE";

String P = "SIMPLE";

// Test Case 2

String T = "AAAAAAAAAAH";

String P = "AAAAH";

// TEST Case 3

String T = "THIS IS MY NEW STRING AAAAHHHH";

String P = "NEW";

// TEST Case 4

String T = "THIS TEST WILL HAVE MULTIPLE MATCHES, SO THAT WE CAN TEST. ONE MORE TEST.";

String P = "TEST";

// TEST Case 5

String T = "COMPUTER SCIENCE IS NO MORE ABOUT COMPUTERS THAN ASTRONOMY IS ABOUT TELESCOPES";

String P = "NO";

// TEST Case 6

String T = "NO COMPUTER IS EVER GOING TO ASK A NEW, REASONABLE QUESTION. IT TAKES TRAINED PEOPLE TO DO THAT.";

String P = "TRAINED";

// TEST Case 7

String T = "WE CAN ONLY SEE A SHORT DISTANCE AHEAD, BUT WE CAN SEE PLENTY THERE THAT NEEDS TO BE DONE.";

String P = "DISTANCE";

// TEST Case 8

String T = "QAZQAZQAZQAZZQAZZQAZZQAZZZZZZZZQQQQQQQZZZZZQAQAQAQAQAZZZQAAAAAZZZQAZZZZZZZQQQQQAAAZZZ";

String P = "QAZZZ";

// TEST Case 9

String T = "COMPUTER SCIENCE IS THE OPERATING SYSTEM FOR ALL INNOVATION.";

String P = "NOVA";

// TEST Case 10

String T = "WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWKWWWWWKWWWWK";

String P = "WWWK";

1. **Number of comparisons (Table format):**

|  |  |
| --- | --- |
| Test Case Number: | Number of Comparisons: |
| 1 | 18 |
| 2 | 35 |
| 3 | 14 |
| 4 | 10 |
| 5 | 23 |
| 6 | 82 |
| 7 | 32 |
| 8 | 47 |
| 9 | 58 |
| 10 | 184 |

### Boyer-Moore-Horspool Algorithm

1. **Description:**
   1. The Boyer-Moore-Horspool Algorithm is a simplified version of the Boyer-Moore Algorithm using a single table (Shift Table). This algorithm preprocesses the pattern to generate a Shift Table to determine the correct mount to shift the pattern when a mismatch occurs. This shift decision is based on the input text character *c’s* alignment with the last character in the pattern.
2. **Data Structures used:**
   1. String
   2. Array
3. **Runtime Analysis:**
   1. Pseudocode

**0** – Algorithm HorspoolMatching(T, P) ---------- --> O(nm) = Worst Case Ex. P = bam-1 and T = an,

**1 - //** O(n/m) = Best Case Ex. P = bm and T = an. However for random texts it is O(n).

**2** - // Input: Pattern P and text T

**3** - // Output: Index the left end of the first matching substring or -1 if no such substring exists

**4** – ShiftTable(P) // Generate Shift Table ---------- --> O( |Σ| + m ) = O( S + m ) // S = Size of Alphabet, m = Size of Pattern.

**5** – i = m -1 // Position of the Pattern’s right end

**6** – while i <= n-1 do ---------- --> O(n)

**7** – k = 0 // Number of matched Characters

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

**9** – k = k + 1

**10** – if k = m

**11** – return i – m + 1

**12** – else i = i + table[T[i]]

**13** – return -1

1. **Sample Input text and patterns:**

// TEST Case 1

String text = "THIS IS A SIMPLE EXAMPLE";

String pattern = "SIMPLE";

// Test Case 2

String text = "AAAAAAAAAAH";

String pattern = "AAAAH";

// TEST Case 3

String text = "THIS IS MY NEW STRING AAAAHHHH";

String pattern = "NEW";

// TEST Case 4

String text = "THIS TEST WILL HAVE MULTIPLE MATCHES, SO THAT WE CAN TEST. ONE MORE TEST.";

String pattern = "TEST";

// TEST Case 5

String text = "COMPUTER SCIENCE IS NO MORE ABOUT COMPUTERS THAN ASTRONOMY IS ABOUT TELESCOPES";

String pattern = "NO";

// TEST Case 6

String text = "NO COMPUTER IS EVER GOING TO ASK A NEW, REASONABLE QUESTION. IT TAKES TRAINED PEOPLE TO DO THAT.";

String pattern = "TRAINED";

// TEST Case 7

String text = "WE CAN ONLY SEE A SHORT DISTANCE AHEAD, BUT WE CAN SEE PLENTY THERE THAT NEEDS TO BE DONE.";

String pattern = "DISTANCE";

// TEST Case 8

String text = "QAZQAZQAZQAZZQAZZQAZZQAZZZZZZZZQQQQQQQZZZZZQAQAQAQAQAZZZQAAAAAZZZQAZZZZZZZQQQQQAAAZZZ";

String pattern = "QAZZZ";

// TEST Case 9

String text = "COMPUTER SCIENCE IS THE OPERATING SYSTEM FOR ALL INNOVATION.";

String pattern = "NOVA";

// TEST Case 10

String text = "WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWKWWWWWKWWWWK";

String pattern = "WWWK";

1. **Number of comparisons (Table format):**

|  |  |
| --- | --- |
| Test Case Number: | Number of Comparisons: |
| 1 | 8 |
| 2 | 11 |
| 3 | 7 |
| 4 | 6 |
| 5 | 14 |
| 6 | 19 |
| 7 | 11 |
| 8 | 11 |
| 9 | 18 |
| 10 | 49 |

### Knuth-Morris-Pratt Algorithm

**Implementation**: [KMP.ipynb - Colaboratory (google.com)](https://colab.research.google.com/drive/1_d04gjSJLokszAEj2NODVgpzjVikZZg7)

https://colab.research.google.com/drive/1\_d04gjSJLokszAEj2NODVgpzjVikZZg7

1. **Description:**

The basic idea behind KMP’s algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text of the next window. We take advantage of this information to avoid matching the characters that we know will anyway match.

1. **Data Structures used:**
   1. String
   2. Array
2. **Runtime Analysis:**

**Time Complexity: O(m+n)**

* 1. Pseudocode

**KMP Algorithm**

**#Longest Prefix Suffix(LPS)**

LPS ← ComputeLPS(Pattern) {build LPS table function}

i ← 0

j ← 0

n ← string length

m ← pattern length

while i < n do

if pattern[j] = string[i] then {if the characters are a match}

i ← i + 1

j ← j + 1

if j = m then {j pointer has reached end of pattern}

return i - j {index of the match}

j ← LPS[j - 1]

else if i<n && pattern[j] != string[i] then {no match}

if j > 0

j ← LPS[j - 1]

else

i ← i + 1

return -1 {no match}

**Longest Prefix Suffix function:**

**LPS ← array [size = pattern length]**

**LPS[0] ← 0 {LPS value of the first element is always 0}**

**len ← 0 {length of previous longest proper prefix that is also a suffix}**

**i ← 1**

**m ← length of pattern**

**while i < m do**

**if pattern[i] is equal to pattern[len] then**

**len ← len + 1**

**LPS[i] ← len**

**i ← i + 1**

**else {pattern[i] is not equal to pattern[len]}**

**if len is not equal to 0 then**

**len ← LPS[len - 1]**

**else {if len is 0}**

**LPS[i] ← 0**

**i ← i + 1**

**return LPS**

1. **Sample Input text and patterns:**

// TEST Case 1

String txt = "THIS IS A SIMPLE EXAMPLE";

String pat = "SIMPLE";

// Test Case 2

String txt = "AAAAAAAAAAH";

String pat = "AAAAH";

// TEST Case 3

String txt = "THIS IS MY NEW STRING AAAAHHHH";

String pat = "NEW";

// TEST Case 4

String txt = "THIS TEST WILL HAVE MULTIPLE MATCHES, SO THAT WE CAN TEST. ONE MORE TEST.";

String pat = "TEST";

// TEST Case 5

String txt = "COMPUTER SCIENCE IS NO MORE ABOUT COMPUTERS THAN ASTRONOMY IS ABOUT TELESCOPES";

String pat = "NO";

// TEST Case 6

String txt = "NO COMPUTER IS EVER GOING TO ASK A NEW, REASONABLE QUESTION. IT TAKES TRAINED PEOPLE TO DO THAT.";

String pat = "TRAINED";

// TEST Case 7

String txt = "WE CAN ONLY SEE A SHORT DISTANCE AHEAD, BUT WE CAN SEE PLENTY THERE THAT NEEDS TO BE DONE.";

String pat = "DISTANCE";

// TEST Case 8

String txt = "QAZQAZQAZQAZZQAZZQAZZQAZZZZZZZZQQQQQQQZZZZZQAQAQAQAQAZZZQAAAAAZZZQAZZZZZZZQQQQQAAAZZZ";

String pat = "QAZZZ";

// TEST Case 9

String txt = "COMPUTER SCIENCE IS THE OPERATING SYSTEM FOR ALL INNOVATION.";

String pat = "NOVA";

// TEST Case 10

String txt = "WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWKWWWWWKWWWWK";

String pat = "WWWK";

1. **Number of comparisons (Table format):**

|  |  |
| --- | --- |
| Test Case Number: | Number of Comparisons: |
| 1 | 19 |
| 2 | 11 |
| 3 | 28 |
| 4 | 74 |
| 5 | 77 |
| 6 | 90 |
| 7 | 83 |
| 8 | 81 |
| 9 | 57 |
| 10 | 60 |