geeksforgeeks

String Matching Algorithms

17/3/2023

Given a text txt[0,1,…, N-1] and a pattern Pat[0,1,…, M-1], write an algorithm search that prints all occurrences of pat[] ~~at~~ in txt[]. you may assume that N ≥ M.

Example: txt[] = "AABAACAADAABAABA"

pat[] = "AABA"

— Pattern found at ~~index~~ indices 0, 9 and 12

~~i = 0 1 2~~ 3

~~j = 0 1~~ M=2

~~N=4, M=2~~

~~4-2~~ = 2

- **Naive Algorithm** (exhaustive search)

Algorithm-Naive (txt [ ], pat [ ])

1. for i = 0 to N - M
2. if Pat[0,1,…M-1] == txt[i, i+1, … i+M-1] ~~then~~
   then print "pattern found"
3.
4. endif
5. end for

complexity: $O(m(n-m))$

- **KMP** (Knuth, Morris, Patterson) Algorithm.

Matching overview

txt = "AAAAABAAABA"

pat = "AAAA"

We compare first window of txt and pat

txt = AAAAABAAABA

pat = AAAA

We find a match (same as naive approach)

In the next step, we compare next window of txt and pat

txt = AAAAABAAABA

pat = AAAA (pattern shifted one position)

This is where KMP does optimization over naive. In this 2nd window, we only compare fourth A of pattern ~~with~~ with fourth character of current window of text to decide whether current window matches or not. Since we know, first 3 characters will anyway match, we skipped matching first 3 characters

Preprocessing :

$lps[i]$ = size of the longest proper prefix of $pat[0,1,\dots,i]$ which is also a suffix of $pat[0,1,\dots,i]$.

- "ABC" → proper prefix are "A", "AB"
  → suffix are "C", "BC", "ABC"

~~pat[] = {A, A, A, A}~~

pat = "AAAA" , lps[] = {0, 1, 2, 3}

pat = "ABCDEF" , lps[] = {0, 0, 0, 0, 0}

pat = "AABAACAABAA", lps[] = {0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5}

pat = "AAACAAAAAC" , lps[] = {0, 1, 2, 0, 1, 2, 3, 3, 3, 4}

Algorithm: lps calculation

1. len ← 0 , lps[0] ← 0

~~for i = 1 to M-1 do~~

while(i < size of pat string)

2. If pat[len] and pat[i] match, then ~~increment len by 1 and assign then~~ ~~len = len + 1~~
3. len = len + 1
4. lps[i] = len, i = i + 1
5. If pat[len] and pat[i] do not match and len > 0 , then
6. ~~len = lps[len]~~ len = lps [len - 1]
   If len = 0 , then i = i + 1
7.

Example: pat = "AAACAAAA" lps[] = {0, 1, 2, 0, 1, 2, 3, 3}

String Matching Algorithms

17/3/2023

P-2

~~1. We start the comparison of pat[j] with j=0 with characters of the current window of text~~

~~2. We keep matching txt[i] and~~

~~2.~~

1. $i \leftarrow 0$, $j \leftarrow 0$
2. If txt[i] and pat[j] match, then
3. $i = i+1$, $j = j+1$

|| for current window

4. If txt[i] and pat[j] do not match
   - we know that pat[0, 1, ..., j-1] match with txt[i-j, ... i-1]
   - We also know lps[j-1] is the count of characters of pat[0, 1, ... j-1] that are both proper prefix and suffix.
   - From the above two points, we can conclude that we do not need to match these lps[j-1] character with ~~txt~~[i-j, ... i-1]

Time: O(N).