CS 225: Data Structures

Homework 5

Group D1

Last Modified on March 23, 2022

Members	
Name	**Student ID**
Li Rong	3200110523
Zhong Tiantian	3200110643
Zhou Ruidi	3200111303
Jiang Wenhan	3200111016

Please turn over for our answers.

Ex. 1 **Our Answer.** (This part is written by Zhong Tiantian)

The list `next[]` must satisfy the following properties by definitions:

(a) `next[0]` $= 0$, since the first element has no prefixes.

(b) `next[`x`] = k` records the largest number k_{max} that k_{max}-prefix $= k_{max}$-postfix. For example, `next[5]` $= 2$ indicates that the 5-th element in `pList[]` is equal to the 2-nd one.

Obviously, this yields that k-prefix is equal to k-postfix.

Assuming that we have known `next[0], next[1], ..., next[`$x-1$`]`, and denote `next[`$x-1$`]` as *now*:

(a) If `pList[`x`]` $=$ `pList[`*now*`]`, which indicates the *now*-th element in `pList[]` is equal to the x-th, then extend the current sublist: `next[`x`]:=next[`$x-1$`]`.

(b) If `pList[`x`]` $\neq$ `pList[`*now*`]`, then go backward by reducing *now* to `next[`*now* -1`]`.

The algorithm is defined below.

Algorithm 1: Build `Next[]`

```
next.append(0)
$x \leftarrow 1$
now $\leftarrow 0$
while $x <$ getLength(pList) do
    if pList[now] $=$ pList[$x$] then
        now $\leftarrow$ now $+ 1$
        $x \leftarrow x + 1$
        call next.append(now)
    end
    else if now $\neq 0$ then
        now $\leftarrow$ next(now $- 1$)
    end
    else
        call next.append(0)
        $x \leftarrow x + 1$
    end
end
```

Time Complexity Analysis In most cases, the algorithm iterate one by one with $x \leftarrow x+1$, so it will perform `Length(pList)` times; in relatively rare cases, where *now* goes backward, the extra time spent can be amortised to the general cases. Thus the amortised time complexity is $O($`length(pList)`$) = O(m)$.

Ex. 2 **Our Answer.** (This part is written by Jiang Wenhan)

Algorithm Description Assume the length of pattern sequence is m.

- Calculate the next sequence of the pattern sequence, which takes time $O(m)$, $m < n$ and align $P[0]$ and $S[0]$.
- Begin comparison between $P[0]$ and $S[0]$, if $P[0] \neq S[0]$.
- Turn back to $S[1]$, and keep doing things above until we found $P[0] = S[k], k \leq n$,

  Once we found such k,

  (a) Align $P[0]$ with $S[k]$ and keep comparing them until we found $P[a] \neq S[k+a]$.
  (b) Right shift the pattern sequence for $m - next[a]$ where front maximal common sublist overlap with the original position of back maximal common sublist.
  (c) Repeat the comparison at this new position (begin with $P[next[a]]$ and $S[k+a]$).

  Whenever we found that $a = m$,

  (a) Record the position k , where the pattern sequence has occur.
  (b) Right shift the pattern sequence by $m - next[m]$ and start new comparison at $P[next[m]]$ and $S[k+m]$.
  (c) Repeat doing this until we reach the end, where the $P[m-1]$ is aligned with $S[n-1]$.

Time Complexity Analysis To calculates its time complexity, we make an assumption of the worst case, where the pattern sequence and target sequence both consists of pairwise different elements. In such case, all elements in *next* sequence are all 0 . The times of shift is $\frac{n}{m}$, and the comparisons for each shift is m , so the time complexity for comparison is $m \times \frac{n}{m}$, which is n. The total time complexity is $O(m) + n$, since m is less equal than n, so the total time complexity is $O(n)$.