String Processing Algorithms 2015 - Week 3 Exercises

Rodion Efremov

November 9, 2015

Exercise 1

Describe how to modify the LSD radix sort algorithm to handle strings of varying length. The time complexity should be the one given in Theorem 1.27.

Solution

The time complexity mentioned in the Theorem 1.27 is $\mathcal{O}(||\mathcal{R}|| + m\sigma)$. All we need to do is to modify the Counting-Sort procedure:

Now, the desired LSD radix sort for variable-length strings is

Exercise 2

Use the lcp comparison technique to modify the standard insertion sort algorithm so that it sorts strings in $\mathcal{O}(\Sigma LCP(\mathcal{R}) + n^2)$ time.

Solution

Why do we set LCP to zero?

Consider the following:

i	S_i	LCP$_{\mathcal{R}}[i]$
1	$aaaba$	0
2	$abaaa$	1
3	aaa	1
4	$baaa$	0
5	$aabba$	0
6	$abaaa$	1

Exercise 3

Give an example showing that the worst case time complexity of string binary search without precomputed lcp information is $\Omega(m \log n)$.

Algorithm 1: COUNTING-SORT($\mathcal{R} = \{S_1, S_2, \ldots, S_n\}, \ell$)

```
for $i = 0$ to $\sigma - 1$ do
    $C[i] = 0$
$s = 0$
for $i = 1$ to $n$ do
    if $|S_i| < \ell$ then
        $s = s + 1$
    else
        $C[S_i[\ell]] = C[S_i[\ell]] + 1$
$sum = s$
for $i = 0$ to $\sigma - 1$ do
    $tmp = C[i]$
    $C[i] = sum$
    $sum = sum + tmp$
$p = 0$
for $i = 1$ to $n$ do
    if $|S_i| < \ell$ then
        $J[p] = S_i$
        $p = p + 1$
    else
        $J[C[S_i[\ell]]] = S_i$
        $C[S_i[\ell]] = C[S_i[\ell]] + 1$
$\mathcal{R} = J$
```

Algorithm 2: LSDRadixSort($\mathcal{R} = \{S_1, S_2, \ldots, S_n\}$)

```
$m = \max_{i=1,2,\ldots,n}\{|S_i|\}$
for $\ell = m$ to $1$ do
    COUNTING-SORT($\mathcal{R}, \ell$)
```

Algorithm 3: INSERTIONSORT($\mathcal{R}, \mathrm{LCP}_{\mathcal{R}}$)

```
for $i = 2$ to $n$ do
    $s = S_i$
    $j = i - 1$
    if $LCP_{\mathcal{R}}[i - 1] = 0$ then
        $\mathrm{LCP}_{\mathcal{R}}[i] = 0$
    while $j > 0$ and LCPCOMPARE($S_j, s, LCP_{\mathcal{R}}[i]$) $> 0$ do
        $S_{j+1} = S_j$
        $j = j - 1$
    $S_{j+1} = s$
```

Solution

Suppose the sorted list of strings is

$$\begin{aligned} &\langle\overbrace{aaa\ldots aaa}^{m}a_1,\\ &\overbrace{aaa\ldots aaa}^{m}a_2,\\ &\qquad\qquad\ldots\\ &\overbrace{aaa\ldots aaa}^{m}a_n\rangle \end{aligned}$$

and the string to search for is $\overbrace{aaa\ldots aaa}^{m}a'$, where $a' \neq a_i$ for any $i \in \{1, 2, \ldots, m\}$ and $a_1 \leq a_2 \leq \cdots \leq a_n$. Now as there is no match, the binary search will do $\Omega(\log n)$ string comparisons, and as we assume the naïve implementation of the string comparison, each comparison will have to iterate through m first characters a before getting to the last character that fails the search, which leads to the worst case time complexity of $\Omega(m \log n)$.

Exercise 4

Let $S[0..n)$ be a string over an integer alphabet. Show how to build a data structure in $\mathcal{O}(n)$ time and space so that afterwards the Karp-Rabin hash function $H(S[i..j))$ for the factor $S[i..j)$ can be computed in constant time for any $0 \leq i \leq j \leq n$.

Solution

We will need the following identity:

$$H(B) = (H(AB) - H(A) \cdot r^{|B|}) \mod q,$$

where

$$H(S[0..n)) = \left(\sum_{i=0}^{n-1} S[i]r^{n-1-i}\right) \mod q.$$

Since we store only the hash values of all the prefixes of the input integer string (and a matrix indexed by the starting and ending indices), we need an efficient way for finding hash values of the factors of the input integer string. If we want to ask for a hash value of a factor B, which is preceded by a factor A, we can compute efficiently $H(B)$ simply by looking up the hashes for $H(A)$ and $H(AB)$. Also we need to cache the values $1, r, r^2, \ldots, r^n$.

Assuming $r = 37, q = 11$ and $S[0..5) = \langle 2, 1, 1, 4, 3\rangle$, we consider all possible prefixes of S:

i	$S[0..i)$	$H(S[0..i))$
0	ε	0
1	$\langle 2\rangle$	2
2	$\langle 2,1\rangle$	9
3	$\langle 2,1,1\rangle$	4
4	$\langle 2,1,1,4\rangle$	9
5	$\langle 2,1,1,4,3\rangle$	6

The actual query routine follows The preprocessing step is

```
Algorithm 4: Query($E, F, i, j, q$)
$l = j - i$
$f = F[l]$
$a = E[j]$
$b = E[i]$
$r = (a - bf) \mod q$
if $r < 0$ then
    return $r + q$
else
    return $r$
```

```
Algorithm 5: Preprocess($S[0..n), r$)
$E[0..n] = [0, 0, \dots, 0]$
$F[0..n] = [0, 0, \dots, 0]$
$h = 0$
for $i = 1$ to $n$ do
    $h = h \times r$
    $h = h + S[i - 1]$
    $E[i] = h$
$f = 1$
for $i = 0$ to $n$ do
    $F[i] = f$
    $f = f * r$
return $(E, F)$
```

Exercise 5

$\Omega(\Sigma LCP(\mathcal{R}))$ is a lower bound for string sorting for any algorithm if characters can be accessed only one at a time. However, for a small alphabet, it is possible to pack several characters into one machine word. Then multiple characters can be accessed simultaneously and treated as if they were a single *super-character*. For example, the string `abbaba` over the alphabet $\Sigma = \{a, b\}$ can be thought of as the string $\{ab, ba, ba\}$ over the alphabet Σ^2. Algorithms taking advantage of this are called *super-alphabet* algorithms.

Develop a super-alphabet version of MSD radix sort. What is the time complexity.

Solution

Algorithm 6: MSDRadixSort($\mathcal{R}, \ell$)

```
if $|\mathcal{R}| < \sigma^2$ then
    return StringQuicksort($\mathcal{R}, \ell$)
$\mathcal{R}_\perp = \{S \in \mathcal{R} \colon |S| = \ell\}$
$\mathcal{R} = \mathcal{R} \setminus \mathcal{R}_\perp$
$(\mathcal{R}_{00}, \mathcal{R}_{01}, \ldots, \mathcal{R}_{0,\sigma-1}, \mathcal{R}_{10}, ..., \mathcal{R}_{\sigma-1,\sigma-1}) = $ CountingSort($\mathcal{R}, \ell$)
for $i = (00)$ to $(\sigma - 1, \sigma - 1)$ do
    MSDRadixSort($\mathcal{R}_i, \ell + 1$)
return $\mathcal{R}_\perp \cdot \mathcal{R}_{00} \cdot \cdots \cdot \mathcal{R}_{\sigma-1,\sigma-1}$
```