Lecture 2: Construction of Suffix Arrays

Johannes Fischer
Normally prefix-doubling algorithms initialize SA for $h = 1$ using a linear-time bucket sort. The main idea [Karp et al. 1972] is as follows:

**Observation 1.** Suppose that $SA_h$ and $ISA_h$ have been computed for some $h > 0$, where $i = SA_h[j]$ is the $j$th suffix in $h$-order and $h$-rank$[i] = ISA_h[i]$. Then, a sort using the integer pairs $(ISA_h[i], ISA_h[i] + h)$ as keys, $i + h \leq n$, computes a $2h$-order of the suffixes.

(Suffixes $i > n - h$ are necessarily already fully ordered.)

The two main prefix-doubling algorithms differ primarily in their application of this observation:

- **Algorithm MM** does an implicit $2h$-sort by performing a left-to-right scan of $SA_h$ that induces the $2h$-rank of $SA_h[j] - h, j = 1, 2, \ldots, n$;
- **Algorithm LS** explicitly sorts each $h$-group using the ternary-split quicksort (TSQS) of Bentley and McIlroy [1993].

**Algorithm MM** employs Observation 1 as follows: If $SA_h$ is scanned left to right (thus, in $h$-order of the suffixes), $j = 1, 2, \ldots, n$, then the suffixes $i - h = SA_h[j] - h > 0$ are necessarily scanned in $2h$-order within their respective $h$-groups in $SA_h$. 

**source:** Puglisi/Smyth/Turpin ACM Computing Surveys ’07
Induced Sorting

- [Nong/Zhang/Chan DCC’09] **sais**-algorithm:
  - ✓ O(n) in theory
  - ✓ fast in practice
  - ✓ as simple as Kärkkäinen/Sanders DC3
Algorithm sais

- Definition: suffix $T[i,n]$ called
  - **S-type** iff $T[i..n] <_{\text{lex}} T[i+1..n]$ ($T[n,n]='$' always S)
  - **L-type** otherwise

1. Choose sample: leftmost S (predecessor is L), $|S^*|<1/2n$
2. Sort $S^*$-suffixes by **recursion**
   - on new text formed by sorted $S^*$-substrings
3. Scan A from left to right (say we’re at pos. $i$):
   - if $T[A[i]-1]$ is L, write $A[i]-1$ to 1st pos. in bucket
4. like (3), but sorting $S$-suffixes in a right-to-left scan
   - if $T[A[i]-1]$ is S, write $A[i]-1$ to last pos. in bucket
\[ T = \begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
c & a & b & c & c & b & a & a & a & b & b & a & \$
\end{array} \]

\begin{array}{cccccccccccccc}
S & S & L & L & L & S^* & S & S & L & L & L & S^* \\
$L & S^*$ & $S$ & $L$ & $L$ & $L$ & $S^*$ & $S$ & $S$ & $L$ & $L$ & $L$ & $S^*$
\end{array}
Sorting $S^*$-Substrings

- Same algorithm, but with UNSORTED $S^*$-suffixes

1. Choose sample: leftmost $S$ (call them $S^*$), $|S^*| < 1/2n$

2. Put $S^*$-substrings in their buckets (in text order)

3. Scan $A$ from left to right (say we’re at pos. $i$):
   - if $T[A[i]-1]$ is L, write $A[i]-1$ to 1st pos. in bucket

4. like (3), but sorting $S$-substrings in a right-to-left scan
Correctness

• 2 main points:
  ▶ S-substrings > L-substrings in same bucket
  ▶ order of suffixes in reduced substring
    \(\triangleq\) order in original string

• full proof: consult section 3.2 in:
  ▶ Ge Nong, Sen Zhang, Wai Hong Chan: 
    *Two Efficient Algorithms for Linear Time Suffix Array Construction.*