Implementation tricks

Finding all maximal repeats

Reducing size of suftab, lcptab and bwttab.

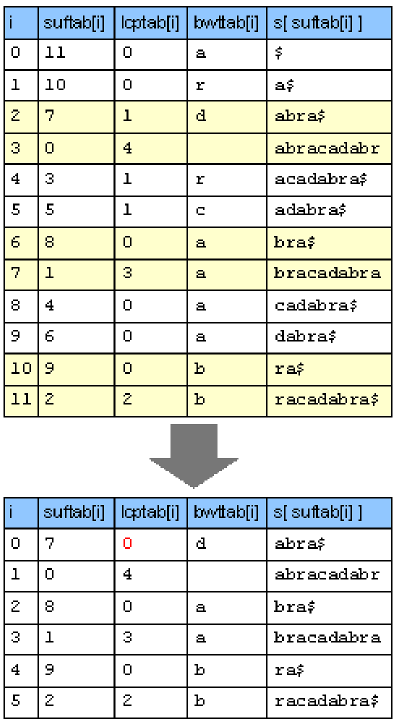
To run a lcp-interval tree algorithm for finding repeats we need to have suftab, lcptab and bwttab available. Together in basic form they occupy 9n bytes, what makes them most memory demanding data structures. Before we go into traversal of lcp-interval tree, which may require additional memory, we can eliminate unnecessary items of the tables.

One important input parameter of algorithm for finding all max. repeats is *m* - minimal length of repeat. When we are interrested only in maximal repeats with length at least m, we may skip processing of l-intervals with $l < m$.

By one simple pass, we reduce suftab, lcptab and bwttab according to lcptab only to those intervals, where lcp is greater than *m*. For sufficiently high values of *m*, size of suftab,lcptab and bwttab may fall considerably. The process is illustrated on fig. 1.

We have to reduce all tables at the same time, so that we can eliminate correct lines also from suftab and bwttab following the information in lcptab.

This new tables represent sequence of subtrees of original lcp-interval tree, rooted at intervals with minimal acceptable lcp value.



i	suftab[i]	lcptab[i]	bwttab[i]	s[suftab[i]]
0	11	0	a	$
1	10	0	r	a$
2	7	1	d	abra$
3	0	4		abracadabr
4	3	1	r	acadabra$
5	5	1	c	adabra$
6	8	0	a	bra$
7	1	3	a	bracadabra
8	4	0	a	cadabra$
9	6	0	a	dabra$
10	9	0	b	ra$
11	2	2	b	racadabra$

i	suftab[i]	lcptab[i]	bwttab[i]	s[suftab[i]]
0	7	0	d	abra$
1	0	4		abracadabr
2	8	0	a	bra$
3	1	3	a	bracadabra
4	9	0	b	ra$
5	2	2	b	racadabra$

Illustration 1: Reduction of suftab, lcptab and bwttab

Note the lcptab[0] value in reduced table. All rows with lcp value lower than *m* were eliminated or lcp value was rewriten to 0. In latter case this value marks boundary of root interval.

Representation of LCP

Since we assume that input is smaller than 2GB (i.e.$n < 2^{31}$) we can represent values in LCP table by 4-byte integer (`int`). So in worst case LCP table takes *4n* bytes.

In most cases LCP values rarely exceed 2^8 or 2^{16} and therefore can be represented by 1- or 2- byte integer (`char` or `short`). When most of the values of LCP table are small, we can save space by using more economical coding and register exceptions for greater values.

Let *m* be minimal match/repeat length. Since we process only items i from *SA* and *LCP* tables where $LCP[i] \geqslant m$, we don't need to represent values smaller than *m*. Furthermore, we need one value to indicate end of root interval and one to indicate exception (value greater than m+254). LCP values will be coded to 1-byte representation according to Table 1. The 2-byte representation is coded similarly.

0	End of interval
1	m
2	m+1
...	
254	m+253
255	Exception

Table 1: 1-byte LCP code

To determine which representation of LCP table to use, we determine following numbers:

k_1 - number of values greater than or equal m+254
k_2 - number of values greater than or equal m+65634
k'_1 - number of values greater than m+254
k'_2 - number of values greater than m+65634

Then, we use rules in Table 2 to decide coding. We start from line 1. If condition in line i holds we use given coding, else we assume conditions 1..i are false and move to line i+1.

i	*condition*	*lcp value coding*	*exception coding*	*table size*
1	$k'_1 = 0$	1-byte	-	n
2	$k'_2 = 0 \wedge n + 2\mathrm{k}_1 < 2\mathrm{n}$	1-byte	2-byte	$n + 2\mathrm{k}_1$
3	$k'_2 = 0$	2-byte	-	$2\mathrm{n}$
4	$n + 4\mathrm{k}_1 < 2\mathrm{n} + 4\mathrm{k}_2$	1-byte	4-byte	$n + 4\mathrm{k}_1$
5	$2\mathrm{n} + 4\mathrm{k}_2 < 4\mathrm{n}$	2-byte	4-byte	$2\mathrm{n} + 4\mathrm{k}_2$
6	true	4-byte	-	$4\mathrm{n}$

Table 2: coding of lcp value