First Huffman, then Burrows-Wheeler: A Simple Alphabet-Independent FM-Index

Szymon Grabowski[1], Veli Mäkinen[2], and Gonzalo Navarro[3]

[1] Computer Engineering Dept., Tech. Univ. of Łódź, Poland.
[2] Dept. of Computer Science, Univ. of Helsinki, Finland.
[3] Dept. of Computer Science, Univ. of Chile, Chile.

Main Results. The basic string matching problem is to determine the occurrences of a short pattern $P = p_1 p_2 \ldots p_m$ in a large text $T = t_1 t_2 \ldots t_n$, over an alphabet of size σ. Indexes are structures built on the text to speed up searches, but they used to take up much space. In recent years, *succinct* text indexes have appeared. A prominent example is the FM-index [2], which takes little space (close to that of the *compressed* text) and *replaces* the text, as it can search the text (in optimal $O(m)$ time) and reproduce any text substring without accessing it. The main problem of the FM-index is that its space usage depends exponentially on σ, that is, $5H_k n + \sigma^\sigma o(n)$ for any k, H_k being the k-th order entropy of T.

In this paper we present a simple variant of the FM-index, which removes its alphabet dependence. We achieve this by, essentially (but not exactly), Huffman-compressing the text and FM-indexing the binary sequence. Our index needs $2n(H_0+1)(1+o(1))$ bits, independent of σ, and it searches in $O(m(H_0+1))$ average time, which can be made $O(m \log \sigma)$ in the worst case. Moreover, our index is considerably simpler to implement than most other succinct indexes.

Technical Details. The Burrows-Wheeler transform (BWT) [1] T^{bwt} of T is a permutation of T such that $T^{bwt}[i]$ is the character preceding the i-th lexicographically smallest suffix of T. The FM-index finds the number of occurrences of P in T by running the following algorithm [2]:

Algorithm FM_Search(P,T^{bwt})
    $i = m$; $sp = 1$; $ep = n$;
    while (($sp \leq ep$) **and** ($i \geq 1$) **do**
        $c = P[i-1]$;
        $sp = C[c] + Occ(T^{bwt}, c, sp-1)+1$;
        $ep = C[c] + Occ(T^{bwt}, c, ep)$;
        $i = i - 1$;
    if ($ep < sp$) **then return** "not found" **else return** "found ($ep - sp + 1$) occs".

The index is actually formed by array $C[\cdot]$, such that $C[c]$ is the number of characters smaller than c in T, and function $Occ(T^{bwt}, \cdot, \cdot)$, such that $Occ(T^{bwt}, c, i)$ is the number of occurrences of c in $T^{bwt}[1 \ldots i]$. The exponential alphabet dependence of the FM-index is incurred in the implementation of Occ in constant time.

Our idea is first to Huffman-compress T so as to obtain T', a binary string of length $n' < n(H_0+1)$. Then, if we encode P to P' with the same codebook used for T, it turns out that any occurrence of P in T is also an occurrence of P' in T' (but not vice versa, as P' may match in the middle of a code in T').

We apply the BWT to T' to obtain array $B = (T')^{bwt}$, of n' bits. Another array Bh signals which bits of B correspond to beginning of codewords in T'. If we apply algorithm FM_Search(P',B), the result is the number of occurrences of P' in T'. Moreover, the algorithm yields the range $[sp, ep]$ of occurrences in B. The real occurrences of P in T correspond to the bits set in $Bh[sp \ldots ep]$.

Function $rank(Bh, i)$, which tells how many bits are set in $Bh[1 \ldots i]$, can be implemented in constant time by storing $o(n')$ bits in addition to Bh [4]. So our number of occurrences is $rank(Bh, ep) - rank(Bh, sp-1)$.

The advantage over the original FM-index is that this time the text T' is binary and thus $Occ(B, 1, i) = rank(B, i)$ and $Occ(B, 0, i) = i - rank(B, i)$. Hence we can implement Occ in constant time using $o(n')$ additional bits, independently of the alphabet size.

Overall we need $2n(H_0 + 1)(1 + o(1))$ bits, and can search for P in $O(m(H_0 + 1))$ time if P distributes as T. By adding $1 + \varepsilon$ bits, for any $\varepsilon > 0$, we can find the text position of each occurrence in worst case time $O((1/\varepsilon)(H_0 + 1)\ \log n)$, and display any text substring of length L in $O((1/\varepsilon)(H_0 + 1)(L + \log n))$ average time. By adding other $2n$ bits, we can ensure that all $O(H_0 + 1)$ values become $O(\log \sigma)$ in the worst case times. For further details and experimental results refer to [3].

References

1. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. *DEC SRC Research Report 124*, 1994.
2. P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. FOCS'00*, pp. 390–398, 2000.
3. Sz. Grabowski, V. Mäkinen and G. Navarro. *First Huffman, then Burrows-Wheeler: A Simple Alphabet-Independent FM-Index.* Technical Report TR/DCC-2004-4, Dept. of Computer Science, University of Chile. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/huffbwt.ps.gz`.
4. I. Munro. Tables. In *Proc. FSTTCS'96*, pp. 37–42, 1996.