Run-Length FM-index

Veli Mäkinen
Department of Computer Science, Univ. of Helsinki, Finland.

Gonzalo Navarro
Department of Computer Science, Univ. of Chile, Chile.

Abstract

The FM-index is a succinct text index needing only $O(H_k n)$ bits of space, where n is the text size and H_k is the kth order entropy of the text. FM-index assumes constant alphabet; it uses exponential space in the alphabet size, σ. In this paper we show how the same ideas can be used to obtain an index needing $O(H_k n)$ bits of space, with the constant factor depending only logarithmically on σ. Our space complexity becomes better as soon as $\sigma \log \sigma > \log n$, which means in practice for all but very small alphabets, even with huge texts. We retain the same search complexity of the FM-index.

FM-index

The FM-index [3] is based on the *Burrows-Wheeler transform (BWT)* [1], which produces a permutation of the original text, denoted by $T^{bwt} = bwt(T)$. String T^{bwt} is a result of the following *forward* transformation: (1) Append to the end of T a special end marker \$, which is lexicographically smaller than any other character; (2) form a *conceptual* matrix $\mathcal{M}$ whose rows are the cyclic shifts of the string $T\$$, sorted in lexicographic order; (3) construct the transformed text L by taking the last column of $\mathcal{M}$. The first column is denoted by F.

The *suffix array* $\mathcal{A}$ of text $T\$$ is essentially the matrix $\mathcal{M}$: $\mathcal{A}[i] = j$ iff the ith row of $\mathcal{M}$ contains string $t_j t_{j+1} \cdots t_n \$ t_1 \cdots t_{j-1}$. Given the suffix array, the search for the occurrences of the pattern $P = p_1 p_2 \cdots p_m$ is trivial. The occurrences form an interval $[sp, ep]$ in $\mathcal{A}$ such that suffixes $t_{\mathcal{A}[i]} t_{\mathcal{A}[i]+1} \cdots t_n$, $sp \leq i \leq ep$, contain the pattern as a prefix. This interval can be searched for using two binary searches in time $O(m \log n)$ [5].

The suffix array of text T is represented implicitly by T^{bwt}. The novel idea of the FM-index is to store T^{bwt} in compressed form, and to simulate a *backward search* in the suffix array as follows:

```
Algorithm FM_Search($P[1,m]$,$T^{bwt}[1,n]$)
$c = P[m]$; $i = m$;
$sp = C_T[c] + 1$; $ep = C_T[c+1]$;
while ($sp \leq ep$) and ($i \geq 2$) do
    $c = P[i-1]$;
    $sp = C_T[c] + Occ(T^{bwt}, c, sp-1)$+1;
    $ep = C_T[c] + Occ(T^{bwt}, c, ep)$;
    $i = i - 1$;
if ($ep < sp$) then return "not found" else return "found ($ep - sp + 1$) occs".
```

The above algorithm finds the interval $[sp, ep]$ of $\mathcal{A}$ containing the occurrences of the pattern P. It uses the array C_T and function $Occ(X, c, i)$, where $C_T[c]$ equals the number of occurrences of characters $\{\$, 1, \ldots, c-1\}$ in the text T and $Occ(X, c, i)$ equals the number of occurrences of character c in the prefix $X[1, i]$.

Ferragina and Manzini [3] go on to describe an implementation of $Occ(T^{bwt}, c, i)$ that uses a compressed form of T^{bwt}; they show how to compute $Occ(T^{bwt}, c, i)$ for any c and i in constant time. However, to achieve this they need exponential space (in the size of the alphabet).

Run-Length FM-Index

Our idea is to exploit run-length compression to represent T^{bwt}. An array S contains one character per run in T^{bwt}, while an array B contains n bits and marks the beginnings of the runs.

Definition 2 *Let string $T^{bwt} = c_1^{\ell_1} c_2^{\ell_2} \ldots c_{n'}^{\ell_{n'}}$ consist of n' runs, so that the i-th run consists of ℓ_i repetitions of character c_i. Our representation of T^{bwt} consists of string $S = c_1 c_2 \ldots c_{n'}$ of length n', and bit array $B = 10^{\ell_1 - 1} 10^{\ell_2 - 1} \ldots 10^{\ell_{n'} - 1}$.*

It is clear that S and B contain enough information to reconstruct T^{bwt}: $T^{bwt}[i] = S[rank(B, i)]$, where $rank(B, i)$ is the number of 1's in $B[1 \ldots i]$ (so $rank(B, 0) = 0$). Function *rank* can be computed in constant time using $o(n)$ extra bits [4, 6, 2]. Hence, S and B give us a representation of T^{bwt} that permits us accessing any character in constant time and requires at most $n' \log \sigma + n + o(n)$ bits. The problem, however, is not only how to access T^{bwt}, but also how to compute $C_T[c] + Occ(T^{bwt}, c, i)$ for any c and i.

In the following we show that the above can be computed by means of a bit array B', obtained by reordering the runs of B in lexicographic order of the characters of each run. Runs of the same character are left in their original order. The use of B' will add $n + o(n)$ bits to our scheme. We also use C_S, which plays the same role of C_T, but it refers to string S.

Definition 3 *Let $S = c_1 c_2 \ldots c_{n'}$ of length n', and $B = 10^{\ell_1 - 1} 10^{\ell_2 - 1} \ldots 10^{\ell_{n'} - 1}$. Let $p_1 p_2 \ldots p_{n'}$ be a permutation of $1 \ldots n'$ such that, for all $1 \leq i < n'$, either $c_{p_i} < c_{p_{i+1}}$ or $c_{p_i} = c_{p_{i+1}}$ and $p_i < p_{i+1}$. Then, bit array B' is defined as $B' = 10^{\ell_{p_1} - 1} 10^{\ell_{p_2} - 1} \ldots 10^{\ell_{p_{n'}} - 1}$.*

We now give the theorems that cover different cases in the computation of $C_T[c] + Occ(T^{bwt}, c, i)$ (see [7] for proofs). They make use of *select*, which is the inverse of *rank*: $select(B', j)$ is the position of the jth 1 in B' (and $select(B', 0) = 0$). Function *select* can be computed in constant time using $o(n)$ extra bits [4, 6, 2].

Theorem 4 *For any $c \in \Sigma$ and $1 \leq i \leq n$, such that $T^{bwt}[i] \neq c$, it holds*

$$C_T[c] + Occ(T^{bwt}, c, i) \quad = \quad select(B', C_S[c] + 1 + Occ(S, c, rank(B, i))) - 1$$

Theorem 5 *For any $c \in \Sigma$ and $1 \leq i \leq n$, such that $T^{bwt}[i] = c$, it holds*

$$\begin{aligned} C_T[c] + Occ(T^{bwt}, c, i) \quad &= \quad select(B', C_S[c] + Occ(S, c, rank(B, i))) \\ &\quad\quad + i - select(B, rank(B, i)). \end{aligned}$$

Since functions $rank$ and $select$ can be computed in constant time, the only obstacle to use the theorems is the computation of Occ over string S.

Instead of representing S explicitly, we will store one bitmap S_c per text character c, so that $S_c[i] = 1$ iff $S[i] = c$. Hence $Occ(S, c, i) = rank(S_c, i)$. It is still possible to determine in constant time whether $T^{bwt}[i] = c$ or not: an equivalent condition is $S_c[rank(B, i)] = 1$.

According to [8], a bit array of length n' where there are f 1's can be represented using $\log \binom{n'}{f} + o(f) + O(\log \log n')$ bits, while still supporting constant time access and constant time $rank$ function for the positions with value 1. It can be shown (see [7]) that the overall size of these structures is at most $n'(\log \sigma + 1.44 + o(1)) + O(\sigma \log n')$.

We have shown in [7] that the number of runs in T^{bwt} is limited by $2H_k n + \sigma^k$. By adding up all our space complexities we obtain $2n(H_k(\log \sigma + 1.44 + o(1)) + 1 + o(1)) + O(\sigma \log n) = 2n(1 + H_k \log \sigma)(1 + o(1))$ bits of space if $\sigma = o(n/\log n)$, for any fixed k.

References

[1] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. *DEC SRC Research Report 124*, 1994.

[2] D. Clark. *Compact Pat Trees.* PhD thesis, University of Waterloo, 1996.

[3] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. FOCS'00*, pp. 390–398, 2000.

[4] G. Jacobson. *Succinct Static Data Structures.* PhD thesis, CMU-CS-89-112, Carnegie Mellon University, 1989.

[5] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22, pp. 935–948, 1993.

[6] I. Munro. Tables. In *Proc. FSTTCS'96*, pp. 37–42, 1996.

[7] V. Mäkinen and G. Navarro. New search algorithms and time/space tradeoffs for succinct suffix arrays. Technical report C-2004-20, Dept. Computer Science, Univ. Helsinki, April 2004.

[8] R. Raman, V. Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. SODA'02*, pp. 233–242, 2002.