Run-Length FM-index (Extended Abstract)

Veli Mäkinen$^1$ and Gonzalo Navarro$^2$

$^1$ Dept. of Computer Science, Univ. of Helsinki, Finland.  
$^2$ Dept. of Computer Science, Univ. of Chile, Chile.

Abstract. The FM-index is a succinct text index needing only $O(H_n n)$ bits of space, where $n$ is the text size and $H_n$ is the $n$th order entropy of the text. Hidden in the sublinear factor lies an exponential dependence on the alphabet size, $\sigma$. In this paper we show how the same ideas can be used to obtain an index needing $O(H_n n)$ bits of space, with the constant factor depending only logarithmically on $\sigma$. 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.

1 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 $T^{\text{but}} = \text{bwt}(T)$. String $T^{\text{but}}$ 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 $M$ 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 $M$. The first column is denoted by $F$.

The suffix array $A$ of text $T\$$ is essentially the matrix $M$: $A[i] = j$ iff the $i$th row of $M$ 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 $[s_{p}, e_{p}]$ in $A$ such that suffixes $t_{A[i]}t_{A[i]+1}\cdots t_{n}$, $s_{p} \leq i \leq e_{p}$, 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 $T^{\text{but}}$. The novel idea of the FM-index is to store $T^{\text{but}}$ in compressed form, and to simulate a backward search in the suffix array as follows:

```
Algorithm FM_Search(P[1,m],T^{\text{but}}[1,n])
(1) c = P[m]; i = m;
(2) s_p = C_T[c] + 1; e_p = C_T[c + 1];
(3) while (s_p \leq e_p) and (i \geq 2) do
  (4) c = P[i-1];
  (5) s_p = C_T[c] + Occ(T^{\text{but}}, c, s_p - 1) + 1;
  (6) e_p = C_T[c] + Occ(T^{\text{but}}, c, e_p);
  (7) i = i - 1;
(8) if (e_p < s_p) then return "not found" else return "found (e_p - s_p + 1) occs".
```

The above algorithm finds the interval $[s_{p}, e_{p}]$ of $A$ 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 $Occ(T^{\text{but}}, c, i)$ that uses a compressed form of $T^{\text{but}}$; they show how to compute $Occ(T^{\text{but}}, c, i)$ for any $c$ and $i$ in constant time. However, to achieve this they need exponential space (in the size of the alphabet).

2 Run-Length FM-Index

Our idea is to exploit run-length compression to represent $T^{\text{but}}$. An array $S$ contains one character per run in $T^{\text{but}}$, while an array $B$ contains $n$ bits and marks the beginnings of the runs.
Definition 1. Let string $T^\text{but} = c_1^i c_2^i \cdots c_n^i$ consist of $n'$ runs, so that the $i$-th run consists of $l_i$ repetitions of character $c_i$. Our representation of $T^\text{but}$ consists of string $S = c_1 c_2 \cdots c_{n'}$ of length $n'$, and bit array $B = 10^{i_1} 10^{i_2} \cdots 10^{i_{n'}}$.

It is clear that $S$ and $B$ contain enough information to reconstruct $T^\text{but}$: $T^\text{but}[i] = S[\text{rank}(B,i)]$, where $\text{rank}(B,i)$ is the number of 1’s in $B[1 \ldots i]$ (so $\text{rank}(B,0) = 0$). Function $\text{rank}$ can be computed in constant time using $o(n)$ extra bits [4,6,2]. Hence, $S$ and $B$ give us a representation of $T^\text{but}$ 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 $T^\text{but}$, but also how to compute $C_T[c] + \text{Occ}(T^\text{but}, c,i)$ 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 2. Let $S = c_1 c_2 \cdots c_{n'}$ of length $n'$, and $B = 10^{i_1} 10^{i_2} \cdots 10^{i_{n'}}$. 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 $B' = 10^{i_1 n'} 10^{i_2 n'} \cdots 10^{i_{n'} n'}$.

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

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

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

Theorem 2. For any $c \in \Sigma$ and $1 \leq i \leq n$, such that $T^\text{but}[i] = c$, it holds

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

Since functions $\text{rank}$ and $\text{select}$ can be computed in constant time, the only obstacle to use the theorems is the computation of $\text{Occ}$ 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 $\text{Occ}(S,c,i) = \text{rank}(S_c,i)$. It is still possible to determine in constant time whether $T^\text{but}[i] = c$ or not: an equivalent condition is $S_c[\text{rank}(B,i)] = 1$.

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 $\text{rank}$ 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 $T^\text{but}$ is limited by $2H_k n + \sigma^k$. By adding up all our space complexities we obtain $2n (H_k (\log \sigma + 1.44) + 1 + o(1)) + O(\sigma \log n) = 2n H_k \log \sigma (1 + o(1))$ bits of space if $\sigma = O(n/\log n)$.

References