Rank/Select on Large Alphabets

Data Structure of Golynski, Munro, and Rao (SODA 2006) [1]

<table>
<thead>
<tr>
<th></th>
<th>$access(T, i)$</th>
<th>$rank(T, i, c)$</th>
<th>$select(T, i, c)$</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>WT</td>
<td>$\log \sigma$</td>
<td>$\log \sigma$</td>
<td>$\log \sigma$</td>
<td>$n \log \sigma + o(n \log \sigma)$</td>
</tr>
<tr>
<td>G-1</td>
<td>$\sigma \cdot \log \log \sigma$</td>
<td>$\log \log \sigma$</td>
<td>1</td>
<td>$nH_0 + O(n)$</td>
</tr>
<tr>
<td>G-2</td>
<td>$\log \log \sigma$</td>
<td>$\log \log \sigma$</td>
<td>1</td>
<td>$n \log \sigma + o(n \log \sigma)$</td>
</tr>
<tr>
<td>G-2a</td>
<td>1</td>
<td>$\log \log \sigma \cdot \log \log \log \sigma$</td>
<td>$\log \log \sigma$</td>
<td>$n \log \sigma + o(n \log \sigma)$</td>
</tr>
</tbody>
</table>
Rank/Select on Large Alphabets

Solution Overview

- (1) Divide sequence into blocks of length $\sigma$
- (2) Solve rank and select on block level
- (3) Solve in-block rank and select
- Step (1) and (2) are used in all structures
Conceptionally introduce a bitvector for each symbol

Concatenated in row major order: Array $A$

Size of $A$: $n\sigma$

```
e y y y m m m m _ _ _ $ e a a r r r r r r r a
$ 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
_ 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
a 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1
e 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
m 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
r 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0
y 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```
Rank/Select on Large Alphabets

If we can answer rank/select on $A$ in constant time, we can answer it on $T$ as well.

\[
\begin{align*}
\text{rank}(T, i, c) &= \text{rank}(A, c \cdot n + i, 1) - \text{rank}(A, c \cdot n, 1) \quad (1) \\
\text{select}(T, i, c) &= \text{select}(A, \text{rank}(A, c \cdot n, 1) + i, 1) \quad (2)
\end{align*}
\]

However, $A$ uses too much space.
Compressing $A$

- Divide $A$ into blocks of length $\sigma$
- Count the number of ones in each block $A$
- Resulting array is $C$ of length $n$, so $|C| = n \log \sigma$ bits.

|   | e | y | y | m | m | m | m | m |   | $\$ |   | e | a | a | r | r | r | r | r | r | a |
| $\$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| _ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 0 | 0 | 1 |
| e | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m | 0 | 0 | 0 | 3 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| r | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 5 | 1 | 1 | 0 | 0 |
| y | 0 | 1 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
Compressing $A$

$C = 010030012110310005300$

- The sum of all entries in $C$ is $n$
- Store it unary-encoded in array $B$ (e.g. $|B| = 2n$ bits)

$B = 1011100011101001011000101111000001000111$

- Add select structure to $B$ to navigate to blocks in $A$
- Jumping to block $i$ in $A$ is $select(B, i, 1)$
- $rank'(A, \sigma i) = rank(B, select(B, i, 1), 0) = select(B, i, 1) + 1 - i$
Rank/Select on Large Alphabets

| e | y | y | y | m | m | m | m | m | m | $ | e | a | a | r | r | r | r | r | a |
| $ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| _ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| e | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| r | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| y | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$B = 101110001110100101011000101111000001000111$

- $\text{rank}'(A, \sigma i) = \text{select}(B, i, 1) + 1 - i$
- $\text{select}'(A, i) = \text{rank}(B, \text{select}(B, i, 0), 1) = \text{select}(B, i, 0) + 1 - i$
Rank/Select on Large Alphabets
In-block rank and select (G-1)

- For each block $A_j$, we store the positions in the range $[0, \sigma - 1]$ of the set bits in increasing order in an array $E_j$
- Total space: $n \log \sigma$

Solve select
Block $x = \text{select}'(A, i, 1)$ contains the $i$-th one. There are $y = \text{rank}'(A, \sigma x, 0)$ ones before block $x$ ⇒
$\text{select}(A, i, 1) = x \cdot \sigma + E_x[i - y]$

Solve rank
$i$ with $j = \left\lfloor \frac{i}{\sigma} \right\rfloor$ and $r = i - j \cdot \sigma$
$\text{rank}(A, i, 1) = \text{rank}'(A, i \cdot \sigma) + \max\{\{k | E_j[k] < r\} \cup \{-1\}\} + 1$
Use $y$-fast trie for second part to get $O(\log \log \sigma)$ time
Rank/Select on Large Alphabets
In-block access, rank, and select (G-2)

- Divide $T$ in chunks of size of size $\sigma$.
- In each chunk $C$: For each $c \in \Sigma$ (in lex. order) write its occurrences in $C$. We get a permutation $\pi$.
- Also store a bitvector $X$ which contains the number of occurrences decoded in unary.

\[
\begin{array}{cccccccccc}
\pi &=& 0 & 4 & 5 & 6 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 6 & 5 & 0 & 0 & 6 & 1 & 2 & 3 & 4 & 5 \\
X &=& 111101000110001010001010101011111001110000001 \\
&   & $._eaemry$ & $._eaemry$ & $._eaemry$ & $._eaemry$ & $._eaemry$
\end{array}
\]
**Rank/Select on Large Alphabets**

In-block access, rank, and select (G-2)

- \(\text{select}(T, i, c)\): First we determine by rank and select on \(A\) chunk \(x\) and the argument \(j\) for select on \(C_x\)

- \(\text{select}(C_x, j, c) = \pi_X[\text{select}(X, c, 1) + j - c]\)

| \(\pi\) | 0 | 4 | 5 | 6 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 6 | 5 | 0 | 0 | 6 | 1 | 2 | 3 | 4 | 5 |

| \(X\) | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| \(X\) | \$ | _ | a | e | m | _ | _ | r | y | \$ | _ | a | e | m | r | y | \$ | _ | a | e | m | r | y | \$ | _ | a | e | m | r | y |
y = \pi^{-1}(i) \text{ tells us the corresponding 0 in } X

- Ones before } y \text{ in } X \text{ the corresponding character}
- I.e. } \text{select}(X, y, 0) = y - 1

\begin{align*}
\pi &= [0, 4, 5, 6, 1, 2, 3, 4, 1, 2, 3, 6, 5, 0, 0, 6, 1, 2, 3, 4, 5] \\
X &= 111101000110001010001010101111110111000001 \\
&\text{$_{a e m r y}$ $\_ a e m r y$ $\_ a e m r y$ $\_ a e m r y$} \\
\end{align*}
Use $X$ to select the range $[sp, ep]$ of position of $c$ in $\pi$.

Solve predecessor query on $\pi[sp..ep]$.

\[\begin{array}{cccccccccccccccc}
e & y & y & y & m & m & m & m & \_ & \_ & \_ & \_ & $ & e & a & a & r & r & r & r & r & r & a \\
\end{array}\]

\[\pi = \begin{bmatrix}
0 & 4 & 5 & 6 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 6 & 5 & 0 & 0 & 6 & 1 & 2 & 3 & 4 & 5
\end{bmatrix}\]

\[\begin{array}{cccccccccccccccc}
X = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\end{array}\]
We already know these bitvector representations

- plain bitvector (*bit_vector*)
- Elias-Fano coded bitvector (*sd_vector*)

In this lecture we will study an $H_0$-compressed bitvector [3].
Let $B$ be a bitvector of length $n$.

\[
H_0(B) = \frac{\kappa}{n} \log \frac{n}{\kappa} + \frac{n - \kappa}{n} \log \frac{n}{n - \kappa},
\]

where $\kappa$ = # of set bits in $B$.

**Theorem [3]**

A bitvector can be represented in $nH_0(B) + o(n)$ bits of space. At the same time rank queries can be performed in constant time.
$H_0$-compressed bitvector

- Split $B$ into block of $K = \frac{1}{2} \log n$ bits
- For each block store the number of set bits (in $\lceil \log K + 1 \rceil$ bits)
- In total these class identifiers sum up to $O\left(n \frac{\log \log n}{\log n}\right)$ bits

- Represent a block as tuple $(\kappa_i, r_i)$, $0 \leq \kappa_i \leq K$ is the class identified and the index $r_i$ within class $\kappa_i$. $r_i \in [0, \binom{K}{\kappa_i} - 1]$.
- The class indexes sum up to

\[
\left\lfloor \log \left( \binom{K}{\kappa_0} \right) \right\rfloor + \cdots + \left\lfloor \log \left( \binom{K}{\kappa_{(n-1)}/K} \right) \right\rfloor \leq \log \left( \binom{K}{\kappa_0} \times \cdots \times \binom{K}{\kappa_{(n-1)}/K} \right) + n/K \\
\leq \log \left( \binom{n}{\kappa_0 + \cdots + \kappa_{(n-1)}/K} \right) + n/K = \log \left( \binom{n}{\kappa} \right) + n/K = nH_0(B) + O(n / \log n)\
\]
$H_0$-compressed bitvector

- Lookup table to map between class indexes and block
- Overall space: $nH_0(B) + O\left(\frac{n}{\log n}\right) + O\left(n\frac{\log \log n}{\log n}\right) = nH_0(B) + o(n)$
- Rank structure: Absolute rank samples + relative rank samples + lookup tables for blocks of size $K = \frac{1}{2} \log n$. 
- Problems in practice:
  - Lookup tables should fit in cache; therefore $K \approx 15$
  - For $K = 15$ class identifiers are not negligible
$H_0$-compressed bitvector

![Graph showing space consumption of $H_0$-compressed bitvectors for different block sizes for two different bitvectors (WEB-wt-1GB and DNA-wt-1GB). The x-axis represents the block size $K$, while the y-axis represents the space in (%) of the original bitvector. The graphs show the space consumption for different components ($\kappa$-array C, $\lambda$-array O, and pointers/samples S) for each block size.](image-url)
On-the-fly block en/de-coding

- Use combinatorial number system of degree $\kappa_i$ to en/de-code a block [2]
- Greedy algorithm is used to en/de-code block
- Required operations:
  - comparison
  - addition/subtraction
On-the-fly block en/de-coding

Figure 5: Encoding of block 100101 into the 5-bit number 13 (5 = d log_6 3 e).

Figure 6: Decoding of tuple (n = 6, bt[2] = 3, btnr[2] = 13) to block 100101.


4 Bitvectors (BV) – bit_vectors.hpp

4.1 rrr_vector
4.2 sd_vector

Given a monotone increasing sequence 0 ≤ x_0 ≤ x_1 ≤ ... ≤ x_m ≤ n. Store the lower `= b log n m` bits explicitly. Store the upper bits as sequence H of unary encoded gaps (0_1 represents k). Use a most 2^m + b log n m bits adding rank and select functionality to H enables constant time access, rank, and select to X.
On-the-fly block en/de-coding

![Diagram showing encoding and decoding process]

Figure 5: Encoding of block 100101 into the 5-bit number 13 (\(5 = \log_2 63\)).

Figure 6: Decoding of tuple \((n = 6, bt[2] = 3, btnr[2] = 13)\) to block 100101.

10
\[v.resize(1048576);\]
11
\[util::set_to_value(v, 5);\]
12
\[cout << size_in_mega_bytes(v) << endl;\]
13
\[// output: 0.375009\]
14
\[int_vector<8> w(1048576, 5);\]
15
\[cout << size_in_mega_bytes(w) << endl;\]
16
\[// output: 1.00001\]


4 Bitvectors (BV) – bit_vectors.hpp

4.1 rrr_vector

4.2 sd_vector

Given a monotone increasing sequence \(0 \leq x_0 \leq x_1 \leq \ldots \leq x_m \leq n\). Store the lower \(\lfloor \frac{b \log n}{m} \rfloor\) bits explicitly. Store the upper bits as sequence \(H\) of unary encoded gaps (\(0^k 1\) represents \(k\)). Use most 2\(m + b \log n\) bits.

Adding rank and select functionality to \(H\) enables constant time access, rank, and select to \(X\).
$H_0$-compressed bitvector

![Graph showing time per operation in $\mu s$ versus block size $K$ for different operations on $H_0$-compressed bitvectors.](image-url)

- bitvector = WEB-WT-1GB
- bitvector = DNA-WT-1GB

- SEL-R$^3K$
- RANK-R$^3K$
- BV-R$^3K$ access

The graph demonstrates the performance of different operations on the $H_0$-compressed bitvector representation for two different bitvectors:WEB-WT-1GB and DNA-WT-1GB. The x-axis represents the block size $K$, and the y-axis represents the time per operation in $\mu s$. The graph shows the time per operation for different values of $K$, and it is clear that the performance of the operations varies with the block size. The SEL-R$^3K$, RANK-R$^3K$, and BV-R$^3K$ access operations are compared, and the results indicate that the performance of these operations is affected by the choice of block size. The graph also shows that the performance of the operations on the DNA-WT-1GB bitvector is generally better than on the WEB-WT-1GB bitvector.
$H_0$ compression for general sequences

Let $S$ be a sequence of length $n$ over alphabet $\Sigma$ of size $\sigma$.

$$H_0(S) = \sum_{\substack{c \in [0, \sigma-1]}} \frac{n_c}{n} \log \frac{n}{n_c}$$

where $n_c$ is the number of occurrences of symbol $c$ in $S$.

Idea
Represent $S$ as a wavelet tree and use $H_0$ to represent the bitvectors.
$H_0$ compression for general sequences

- $B_{\text{root}}[0, n-1]$ has $n_0$ 0s and $n_1$ 1s, then the first level requires
  \[ n_0 \log \frac{n}{n_0} + n_1 \log \frac{n}{n_1} \text{ bits} \]

- Say $v_0$ is the left and $v_1$ the right child of the root.
- And $B_{v_0}[0, n_0 - 1]$ has $n_{00}$ 0s and $n_{01}$ 1s.
- Then $B_{v_0}$ uses
  \[ n_{00} \log \frac{n_0}{n_{00}} + n_{01} \log \frac{n_0}{n_{01}} \text{ bits} \]

- Similarly, $N_{v_1}$, uses
  \[ n_{10} \log \frac{n_1}{n_{10}} + n_{11} \log \frac{n_1}{n_{11}} \text{ bits} \]
**$H_0$ compression for general sequences**

- Summing up the space for the three bitmaps results in

$$n_{00} \log \frac{n}{n_{00}} + n_{01} \log \frac{n}{n_{01}} + n_{10} \log \frac{n}{n_{10}} + n_{11} \log \frac{n}{n_{11}} \text{ bits}$$

- I.e. $nH_0(S)$ for $\sigma = 4$.
- In general the sum of all $n_v H(B_v)$ is $nH_0(S)$.

Since rank and select can be answered in constant time we get

- $O(\log \sigma)$ access
- $O(\log \sigma)$ rank
- $O(\log \sigma)$ select
$H_k$ compression

Figure 6.1: Calculation of $H_1(T)$. The green nodes correspond to the contexts.

$H_1(T) = 16(5H_0([1,4]) + 6H_0([2,3,1])) = 16(5(15\log_5 4 + 4\log_5 4) + 6(13\log_3 4 + 12\log_3 6)) = 7.773$.

Note that it is also possible to use a bottom-up traversal of the CST to calculate $H_k$ in linear time. However, by using a depth-first-search order traversal we can skip many subtrees (cf. line 9 in Listing 6.1) if $k$ is small. This is not possible in the bottom-up traversal.
Rank/select operations on large alphabets: a tool for text indexing. 

Gonzalo Navarro and Eliana Providel. 
Fast, small, simple rank/select on bitmaps. 

Succinct indexable dictionaries with applications to encoding k-ary trees and multiset. 