Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Dr. Simon Gog

2. Project in Text Indexing (WS 2016/17)

http://algo2.iti.kit.edu/3175.php
gog@kit.edu

Exercise 1 (*More space-efficient SA sampling*)

The bitvector which was used in the lecture to indicate whether $SA[i] \equiv 0 \bmod s$ is a considerable overhead for texts over small alphabets like DNA sequences, especially if it is not compressed. Early CSA implementations use a much simpler scheme. The scheme samples every i-th value in SA (i.e. $SA[i]$ is sampled iff $i \equiv 0 \bmod s$). We call the latter scheme *SA-order-sampling* and the first one *text-order-sampling*.

- Does the upper bound of $s - 1$ LF steps to calculate an arbitrary SA value still hold in the new scheme? Proof your answer.
- Present an example where *text-order-sampling* is faster in practice than *SA-order-sampling* and vice versa.

Exercise 2 (*Combinatorial numeral system*)

Use the bijection presented in the lecture to transform the following bitvectors of length $L = 6$ with κ_i set bits to unique numbers λ_i in $[0, \binom{L}{\kappa_i})$

- `000000`
- `100110`,
- `001000`

Apply the reverse transformation to reconstruct the bitvectors of length $L = 6$ for the following (κ_i, λ_i)-pairs:

- $(5, 1)$
- $(1, 1)$
- $(3, 17)$

Exercise 3 (H_0-*compressed bitvector*)

In the lecture we have discussed the compressed bitvector of Raman, Raman, and Rao (`rrr_vector<>` in SDSL). We have seen that increasing the block size K decreased the size of the `rrr_vector` representation of different bitvectors.
Is this true in general? Hint: Try to find a counter example.

Exercise 4 (*„2 in 1"*)

We have seen that Ψ and LF can be expressed via SA and ISA. Therefore a naive implementation of a data structure which supports constant time access to Ψ, LF, SA and ISA takes $2n \log n$ bits. Give a solution which takes only $n \log n + o(n \log n)$ bits and still provides access in $O(\log n)$ time.

- Implement your solution.
- Extra point: Integrate your implementation in the SDSL framework.