**Linear Work Suffix Array Construction**

Juha Karkkainen, Peter Sanders, Stefan Burkhardt

*COMP7295/8295 Bioinformatics Algorithm*

*Presenter: Nabin Maharjan, and Quang Tran*

The algorithm is called DC3 (for Difference Cover modulo 3). This linear-time suffix array construction algorithm following the structure of Farach’s algorithm[1] but using 2/3-recursion instead of half-recursion has been proposed in this paper.

1. Construct the suffix array of the suffixes starting at positions i mod 3 ≠ 0. This is done by reduction to the suffix array construction of a string of two thirds the length, which is solved recursively.
2. Construct the suffix array of the remaining suffixes using the result of the first step.
3. Merge the two suffix arrays into one.

The execution of this algorithm is explained by the following example:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **y** | **a** | **b** | **b** | **a** | **d** | **a** | **b** | **b** | **a** | **d** | **o** |

Where the result we expect to get:

SA = (12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0)

**Step 0: Construct a sample**

Bk = {i ∈ [0, n] | i mod 3 = k}, k = 0, 1, 2

Let C = B1 ∪ B2 be the set of sample positions and SC the set of sample suffixes

B1 = {1, 4, 7, 10}, B2 = {2, 5, 8, 11}, i.e., C = {1, 4, 7, 10, 2, 5, 8, 11}

**Step 1: Sort sample suffixes**

Rk = [tktk+1tk+2][tk+3tk+4tk+5] . . . [tmax Bk tmax Bk+1tmax Bk+2], k = 1, 2

Whose characters are triples [titi+1ti+2]. Note that the last character of Rk is always unique because tmaxBk+2 = 0. Let R = R1ΘR2 be the concatenation of R1 and R2. Then the (nonempty) suffixes of R correspond to the set SC of sample suffixes: [titi+1ti+2][ti+3ti+4ti+5] . . . corresponds to Si . The correspondence is order preserving, i.e., by sorting the suffixes of R we get the order of the sample suffixes SC.  
R = [abb][ada][bba][do0][bba][dab][bad][o00]

To sort the suffixes of R, first radix sort the characters of R and rename them with their ranks to obtain the string R’. If all characters are different, the order of characters gives directly the order of suffixes. Otherwise, sort the suffixes of R’ using Algorithm DC3.

R’ = (1, 2, 4, 6, 4, 5, 3, 7) and SAR’ = (8, 0, 1, 6, 4, 2, 5, 3, 7)

Once the sample suffixes are sorted, assign a rank to each suffix. For i ∈ C, let rank(Si) denote the rank of Si in the sample set SC. Additionally, define rank(Sn+1) = rank(Sn+2) = 0. For i ∈ B0, rank(Si) is undefined.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| **rank(Si)** | **╩** | **1** | **4** | **╩** | **2** | **6** | **╩** | **5** | **3** | **╩** | **7** | **8** | **╩** | **0** | **0** |

**Step 2: Sort nonsample suffixes**

Represent each nonsample suffix Si ∈ SBo with the pair (ti , rank(Si+1)). Note that rank(Si+1) is always defined for i ∈ B0. Clearly we have, for all i, j ∈ B0, Si ≤ Sj ⇐⇒ (ti , rank(Si+1)) ≤ (tj , rank(Sj+1)). The pairs (ti , rank(Si+1)) are then radix sorted.

S12 < S6 < S9 < S3 < S0 because (0, 0) < (a, 5) < (a, 7) < (b, 2) < (y, 1)

**Step 3: Merge**

The two sorted sets of suffixes are merged using a standard comparison-based merging. To compare suffix Si ∈ SC with Sj ∈ SBo , we distinguish two cases:

i ∈ B1 : Si ≤ Sj ⇐⇒ (ti , rank(Si+1)) ≤ (tj , rank(Sj+1))

i ∈ B2 : Si ≤ Sj ⇐⇒ (ti ,ti+1, rank(Si+2)) ≤ (tj ,tj+1, rank(Sj+2))

Note that the ranks are defined in all cases.

S1 < S6 because (a, 4) < (a, 5) and S3 < S8 because (b, a, 6) < (b, a, 7)

**The time complexity of Algorithm DC3:** T(n) = T(2n/3) + O(n) Є O(n)

[1] M. Farach. Optimal suffix tree construction with large alphabets. In Proc. 38th Annual Symposium on Foundations of Computer Science, pages 137–143. IEEE, 1997