

DC3 Algorithm Tutorial

Tutorial By

Vikas Awadhiya

This document is licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/)

This document is licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/) (Creative Commons Attribution 4.0 International License)



DC3 Algorithm Invented/Developed By,

Juha Kärkkäinen

&

Peter Sanders

Tutorial By

Vikas Awadhiya

LinkedIn Profile: <https://in.linkedin.com/in/awadhiya-vikas>

Introduction

DC3 is recursive linear time construction algorithm of suffix array. Suffix array is a simple array contains starting index of all suffixes of a string (for which suffix array is created) in such way that all suffixes are in lexicographical sorted order.

Algorithm internally uses radix sort which is efficient sorting technique for strings. Input string is converted into number array by converting each characters of string to number which make it easier to apply radix sort. Then algorithm divides suffixes into two groups and sorts each group individually. By dividing suffixes in two groups, algorithm uses one group to sort other group and avoid same comparison again and again. In final step merges those groups to find suffix array.

Algorithm

DC3 Algorithm can be viewed as 4 steps. These steps are as follows,

1. Create numeric array by converting each character of input string to number and finally append three zeros. Zero is sentinel number in algorithm, few tutorials uses \$ as sentinel character but we will use zero as sentinel number (0 represent NUL character in ASCII table)
2. Find R_{12} orders, here R_{12} represent sorted orders of all suffixes start at indexes, such that indexes having remainder 1 or 2 when mod by 3. It means, $1 == \text{index} \% 3$ or $2 == \text{index} \% 3$. This step may require recursive call. Order of R_{12} is required to find before R_0 order.
3. Find R_0 orders with the help of R_{12} orders, here R_0 represent sorted orders of all suffixes start at indexes such that indexes having remainder 0 when mod by 3. It means $0 == \text{index} \% 3$.
4. Merge R_0 and R_{12} orders, the merge result is suffix array.

An example is required to explain these steps, so let's consider a string,

S = abcabcacab

Before proceeding let's see the suffix array of S,

S	=	a	b	c	a	b	c	a	c	a	b
Index =>		0	1	2	3	4	5	6	7	8	9

Fig 1.0

Suffix array = [8, 0, 3, 6, 9, 1, 4, 7, 2, 5]

It would be more illustrative if suffix starting at respective index is also written as follows,

Suffix Array	Suffix
8	ab
0	abcabcacab
3	abcacab
6	acab
9	b
1	bcabcacab
4	bcacab
7	cab
2	cabcacab
5	cacab

Fig 2.0

Note: Above example doesn't include entry of sentinel character in suffix array, most of tutorials have element of sentinel character \$ as first element in suffix array but this document will not.

After above example it is clear how suffix array represent all suffixes of string S in lexicographical sorted order. Suffix array contains starting index of all suffixes. Now let's see how it can be created step by steps. Algorithm begins by converting each character of string S to numbers.

Converting each char to number

First step of algorithm is to convert each character of string S into number and append three zeros at end. A character is converted to its ASCII table numeric value. So character 'a' will be converted to 97, 'b' and 'c' shall be converted to 98, 99 respectively.

Characters to number conversion then append three zeros,

a	b	c	a	b	c	a	c	a	b			
97	98	99	97	98	99	97	99	97	98	0	0	0

Fig 3.0

T = [97, 98, 99, 97, 98, 99, 97, 99, 97, 98, 0, 0, 0]

After converting characters to numbers algorithm will not use input string S and will use numeric array T created after conversion. Now next step is to find order of R_{12} ,

R_{12} Order

The R_{12} order is lexicographical sorted order of all suffixes of string S starting at indexes such that remainder of mod by 3 of index is either one or two ($1 == \text{index} \% 3$ or $2 == \text{index} \% 3$). Following are R_{12} indexes,

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9	10	11	12

Fig 4.0

As show above in fig 4.0 all indexes having property $1 == \text{index} \% 3$ are orange underlined and indexes having property $2 == \text{index} \% 3$ are blue underlined. R_{12} indexes, first include indexes having property $1 == \text{index} \% 3$ and then indexes having property $2 == \text{index} \% 3$ as follows,

R_{12} Indexes = [1, 4, 7, 10, 2, 5, 8]

Note: Algorithm doesn't include last two indexes of T, in this case 11 and 12, in R_{12} or R_0 indexes, because sequence of three numbers cannot be created from last two indexes.

To find the R_{12} order, algorithm creates sequences of three numbers, starting from R_{12} indexes as follows,

R_{12} Indexes = [1, 4, 7, 10, 2, 5, 8]

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9	10	11	12

R_{12} = [98, 99, 97], [98, 99, 97], [99, 97, 98], [0, 0, 0], [99, 97, 98], [99, 97, 99], [97, 98, 0]

Index => 1 4 7 10 2 5 8

Fig 5.0

Here index in R_{12} in fig 5.0 represent starting index of sequence.

To understand what is happening with respect to original string, you can view numbers to their character representation,

R_{12} = [bca], [bca], [cab], [NUL, NUL, NUL], [cab], [cac], [ab, NUL]

But algorithm always uses numeric sequences of R_{12} show above in fig 5.0

Sequences are not in sorted order, algorithm uses radix sort in a manner similar to most significant digit first to sort them.

R_{12} = [98, 99, 97], [98, 99, 97], [99, 97, 98], [0, 0, 0], [99, 97, 98], [99, 97, 99], [97, 98, 0]

First digits of these sequences highlighted by underline shall be compared by radix sort, and following shall be the result after first round of sorting,

R_{12} = [0, 0, 0], [97, 98, 0], [98, 99, 97], [98, 99, 97], [99, 97, 98], [99, 97, 98], [99, 97, 99]

Index => 10 8 1 4 7 2 5

First round of radix sort is not enough to sort these sequences. Sequences are divided in 4 buckets (buckets are highlighting by different underline colors) and two of these buckets having more than one sequence. Another attempt will be made to sort these sequence, now this time radix sort will use second number to sort individual buckets separately as follows,

[98, 99, 97], [98, 99, 97]

1 4

[99, 97, 98], [99, 97, 98], [99, 97, 99]

7 2 5

But there would be no change by sorting using second number in both buckets because these second numbers are same among all the sequences in same buckets. So one last attempt will be made to sort these sequences, this time radix sort will use third number to sort these sequences.

[98, 99, 97], [98, 99, 97]

1 4

[99, 97, 98], [99, 97, 98], [99, 97, 99]

7 2 5

As indicated by underlining, sequences start at index 1 and 4 having same third number and this last round of sort will not have any effect, but bucket having sequences of starting indexes 7, 2, and 5 will be split in two buckets because $98 < 99$. So final result after radix sort completion on R_{12} as follows,

$R_{12} =$ [0, 0, 0], [97, 98, 0], [98, 99, 97], [98, 99, 97], [99, 97, 98], [99, 97, 98], [99, 97, 99]

Index => 10 8 1 4 7 2 5

Radix sort cannot find unique order of R_{12} , because few sequences are equal. But after sorting some order has been established. So if sequences are ordered base on their bucket few sequences shall get unique order and few sequences shall share order with sequences are equal or are in same bucket.

Note: Order always starts from 1 and not from 0, because zero is reserved as sentinel number by algorithm itself.

Elements shall be ordered buckets-wise as follows,

Order => 1 2 3 3 4 4 5
 $R_{12} =$ [0, 0, 0], [97, 98, 0], [98, 99, 97], [98, 99, 97], [99, 97, 98], [99, 97, 98], [99, 97, 99]

Index => 10 8 1 4 7 2 5

Fig 6.0

As listed above in fig 6.0, sequences start at index 1 and 4 having same order 3, and sequences start at index 7 and 2 having same order 4.

Now algorithm will use recursive call to find R_{12} order because it is not possible to find unique order of R_{12} using radix sort. But to make a recursive call something similar to T is required and that would be T` (single dash because it is first level of recursion). But before to look into T` creation, first see what is known and what can be done other than recursion.

In R_{12} order few sequences have same order because they are equal. But suffixes start from same indexes from where these equal sequences start, may have more than three numbers. So

order among these equal sequences can be found by comparing numbers from fourth number onward (because first three numbers are already compared and they are equal). But you will find fourth number or numbers onward from it belongs to some sequence and order of that sequence is already known. It means number wise comparison is not required. Order (not unique order) of all sequences of three numbers start from R_{12} indexes is known and R_{12} order can be used to replace three numbers sequences in all suffixes start at R_{12} indexes and after that, these suffixes can be sort in less number of comparisons to find R_{12} order. For example sequence start from index 1 and index 4 are equal and have same order in R_{12} , so number sequences of suffixes start from index 1 and index 4 can be replaced and order among them can be found as follows,

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9	10	11	12

Order =>	1	2	3	3	4	4	5
R_{12} =	<u>[0, 0, 0]</u>	<u>[97, 98, 0]</u>	<u>[98, 99, 97]</u>	<u>[98, 99, 97]</u>	<u>[99, 97, 98]</u>	<u>[99, 97, 98]</u>	<u>[99, 97, 99]</u>
Index =>	10	8	1	4	7	2	5

$$\begin{aligned} \text{Suffix}_1 &= \underline{98, 99, 97}, \underline{98, 99, 97}, \underline{99, 97, 98}, \underline{0, 0, 0} \\ &\quad \quad \quad 3 \quad \quad \quad 3 \quad \quad \quad 4 \quad \quad \quad 1 \\ &= 3, 3, 4, 1 \end{aligned}$$

$$\begin{aligned} \text{Suffix}_4 &= \underline{98, 99, 97}, \underline{99, 97, 98}, \underline{0, 0, 0} \\ &\quad \quad \quad 3 \quad \quad \quad 4 \quad \quad \quad 1 \\ &= 3, 4, 1 \end{aligned}$$

After sequences are replaced by R_{12} orders, order among suffix_1 and suffix_4 can be found in less number of comparisons. Same can be done for other suffixes and unique order of R_{12} can be found. But rather than doing it on various suffixes, this can be done on all sequences start on R_{12} indexes in T as follows,

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9	10	11	12

As highlighted in purple and green underline in T all sequences start at indexes having property $1 == \text{index} \% 3$ are replaced by R_{12} order. These sequences start from indexes 1, 4, 7 and 10.

$$= 3, 3, 4, 1$$

Then all sequences start at indexes having property $2 == \text{index} \% 3$ are replaced by R_{12} order. These sequences start from indexes 2, 5 and 8 as highlighted in purple and green underline in T below,

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9	10	11	12

= 4, 5, 2

And when you combine these they are called T' , So T' is

$T' = [3, 3, 4, 1, 4, 5, 2]$

Now finding suffix array of T' is equivalent of finding R_{12} order.

All theory present above to explain concept behind T' and to explain what is the origin of T' but in practices T' can be created quickly with the help of R_{12} order and R_{12} indexes are follows,

Order =>	1	2	3	3	4	4	5
$R_{12} =$	<u>[0, 0, 0]</u>	<u>[97, 98, 0]</u>	<u>[98, 99, 97]</u>	<u>[98, 99, 97]</u>	<u>[99, 97, 98]</u>	<u>[99, 97, 98]</u>	<u>[99, 97, 99]</u>
Index =>	10	8	1	4	7	2	5

R_{12} Indexes = [1, 4, 7, 10, 2, 5, 8]

As you can see, first element of R_{12} indexes is 1 and sequence start from index 1 have order 3 in R_{12} order, so 3 will be inserted in T'

$T' = [3,$

Then, second entry of R_{12} indexes is 4 and sequence start from index 4 have order 3 in R_{12} order, so 3 will be inserted,

$T' = [3, 3,$

Similar for all elements of R_{12} indexes, and T' would be,

$T' = [3, 3, 4, 1, 4, 5, 2]$

You may have notice that, this is the reason why R_{12} indexes first include all indexes having property $1 == \text{index} \% 3$ and then indexes having property $2 == \text{index} \% 3$, and due to this sequences of numbers are preserved when creating T' from T.

One last thing remains before T' can be used for recursive call and that is, appending three zeros. So T' would be,

$T' =$

3	3	4	1	4	5	2	0	0	0
---	---	---	---	---	---	---	---	---	---

Now algorithm recursively calls itself to find suffix array of T' . Suffix array of T' will be remapped to R_{12} order. Recursive call starts from step 2 because T' is already an array of numbers not a string, where every character has to be converted to number.

R'_{12} Order

Procedure of finding R_{12} order is already explained and doesn't require to explaining again while finding R'_{12} order.

$T' =$

3	3	4	1	4	5	2	0	0	0
---	---	---	---	---	---	---	---	---	---

Index => 0 1 2 3 4 5 6 7 8 9

R'_{12} Indexes = [1, 4, 7, 2, 5]

$R'_{12} =$ [3, 4, 1], [4, 5, 2], [0, 0, 0], [4, 1, 4], [5, 2, 0]

Index => 1 4 7 2 5

Fig 7.0

To find the order, algorithm tries to sort R'_{12} using radix sort,

Sorting by first number,

$R'_{12} =$ [3, 4, 1], [4, 5, 2], [0, 0, 0], [4, 1, 4], [5, 2, 0]

After first round of sorting,

$R'_{12} =$ [0, 0, 0], [3, 4, 1], [4, 5, 2], [4, 1, 4], [5, 2, 0]
Index => 7 1 4 2 5

Fig 8.0

As highlighted by underline with different colors, above in fig 8.0, only one bucket having two sequences. Radix sort attempts again to sort only this bucket. Radix sort will use second number this time,

[4, 5, 2], [4, 1, 4]
 4 2

After second round of radix sort no two or more sequences are equal and each sequence is either less than or greater than other sequences, it means unique order is found and order is as follows,

$R'_{12} =$ [0, 0, 0], [3, 4, 1], [4, 1, 4], [4, 5, 2], [5, 2, 0]
Index => 7 1 2 4 5

Fig 9.0

So radix sort is able to sort R_{12} sequences in such way that all of the sequences have unique order and further recursion is not required. Now R_{12} is written in sorted order. Unique order of R_{12} is found and writing sequences of three numbers are not required. Rather sequences can be replaced by the starting index of these sequences as follows,

$$R_{12} = [7, 1, 2, 4, 5]$$

Next step is to find R_0 order. To finding R_0 order, requires R_{12} order and due to this R_{12} was evaluated first.

R_0 Order

R_0 order is lexicographical sorted order of all suffixes, start at indexes such that indexes have property, $0 == \text{index} \% 3$,

$T =$	3	3	4	1	4	5	2	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9

Fig 10.0

R_0 indexes are highlighted by orange underline in fig 10.0 above.

$$R_0 \text{ Indexes} = [0, 3, 6]$$

A different technique is used to sort R_0 as compared to R_{12} . In R_{12} sequence of three continuous numbers starting from R_{12} indexes were created but to find R_0 order, pairs are created for each R_0 index. First element of each pair is value at index and second element is index itself. Following shall be the pairs,

$$R_0 = [3, 0], [1, 3], [2, 6]$$

R_0 have three pairs only, first pair is $[3, 0]$ here first element is value at index that is $T[0]$ and second element is index itself. Similarly next pair have $T[3]$ and index 3 and last pair have $T[6]$, and index 6.

Radix sort can perform maximum two rounds to find R_0 order but in both round radix sort will compare first element of pair, not similar to R_{12} sorting procedure. R_{12} order is already known and with the help of R_{12} order, radix sort is capable to find the R_0 order. This step does not trigger recursion. In this example finding R_0 order of T will not require second round of radix sort, single round of sort will produce unique order but you will see the second round in work later in this document.

$$R_0 = [\underline{3}, 0], [\underline{1}, 3], [\underline{2}, 6]$$

As highlighted by underline above radix sort will sort by first element of each pair. As you can see all the numbers 3, 1 and 2 are different and no number is equal to other number and due to this after first round of sort, unique order of R'_0 will be found as follows,

$R'_0 = [1, 3], [2, 6], [3, 0]$

R'_0 order is found, now R'_0 is written in sorted order and these pair can be replace by their respective index (second element of pair) and R'_0 finally would be,

$R'_0 = [3, 6, 0]$

Order of R'_0 and R'_{12} is found. To find suffix array of T' requires merging R'_0 and R'_{12} orders.

Merging R'_0 and R'_{12} orders

Merging of R'_0 and R'_{12} is similar to merging of two integer arrays in one big array but merging also take advantage of knowledge of R'_{12} order as follows,

$T' =$	3	3	4	1	4	5	2	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9

$R'_0 = 3, 6, 0$



$R'_{12} = 7, 1, 2, 4, 5$



Fig 11.0

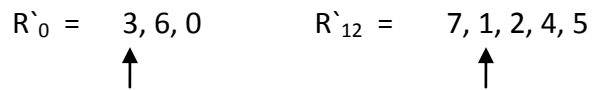
As you can see in fig 11.0 above, merging starts from first entry of both R'_0 and R'_{12} orders. Let $a = 3$ and $b = 7$, here a and b can be any value in R'_0 and R'_{12} respectively. Comparing $T'[a]$ and $T'[b]$, here three possibilities are as follows,

1. $T'[a] < T'[b]$ or $T'[b] < T'[a]$, It doesn't matter which one is smaller but they are not equal and due to this no further process is required, small value will be merged.
2. $T'[a] == T'[b]$, and $1 == b \% 3$, next numbers $T'[a+1]$ and $T'[b+1]$ should be compared, but number wise comparison is not required in this case, because $1 == b \% 3$ and b will continue belong to R'_{12} after adding one, $2 == (b + 1) \% 3$. But after adding one in a , a will no longer belong to R'_0 but it will belong to R'_{12} , $1 == (a + 1) \% 3$. So both next indexes belong to R'_{12} and no number wise comparison is required because R'_{12} order is already known.
3. $T'[a] == T'[b]$, and $2 == b \% 3$, in this case next numbers have to be compared. If $T'[a+1]$ and $T'[b+1]$ are equal again then no further number wise comparison is required. $2 == (a + 2) \% 3$ and $1 == (b + 2) \% 3$, both $a+2$ and $b+2$ indexes belong to R'_{12} and order of R'_{12} is already known.

Let's merge R'_0 and R'_{12} ,

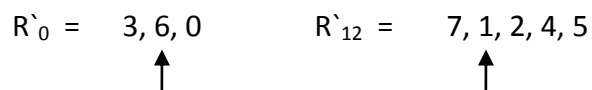
$T'[3]$ is 1 and $T'[7]$ is 0. $T'[7] < T'[3]$. So 7 will be merged,

T' Suffix array = [7,

$R'_0 = 3, 6, 0$ $R'_{12} = 7, 1, 2, 4, 5$


Then, $T'[3]$ and $T'[1]$ shall be compared as highlighted by pointing arrow, $T'[3]$ is 1 and $T'[1]$ is 3. $T'[3] < T'[1]$, So 3 will be merged,

T' Suffix array = [7, 3,

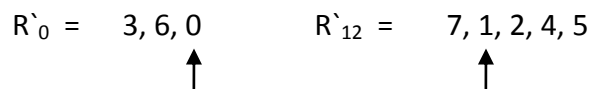
$R'_0 = 3, 6, 0$ $R'_{12} = 7, 1, 2, 4, 5$


Then, $T'[6]$ and $T'[1]$ shall be compared, $T'[6]$ is 2 and $T'[1]$ is 3. $T'[6] < T'[1]$, So 6 will be merged,

T' Suffix array = [7, 3, 6

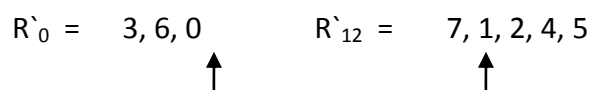
$T' =$	3	3	4	1	4	5	2	0	0	0
Index=>	0	1	2	3	4	5	6	7	8	9

R'_{12}	7	1	2	4	5
Index=>	0	1	2	3	4

$R'_0 = 3, 6, 0$ $R'_{12} = 7, 1, 2, 4, 5$


Finally $T'[0]$ and $T'[1]$ shall be compared, but $T'[0] == T'[1]$, both are equal. Here number wise comparison is not required because $1 == (0 + 1) \% 3$ and $2 == (1 + 1) \% 3$ and both belong to R'_{12} . As you can see above in R'_{12} order, 1 is at position 1 and 2 is at position 2. So 1 comes before 2 in R'_{12} , it means order of 1 is less than order of 2 or suffix start at index 1 is less than suffix start at index 2, so 0 will be merged.

T' Suffix array = [7, 3, 6, 0,

$R'_0 = 3, 6, 0$ $R'_{12} = 7, 1, 2, 4, 5$


After merging 0, R'_0 order is exhausted and all remaining elements in R'_{12} shall be merged at end as follows,

T` Suffix array = [7, 3, 6, 0, 1, 2, 4, 5]

The first entry of T` suffix array is 7 and that is entry of sentinel number, this sentinel number is not a part of original input T` but later on added by algorithm to construct suffix array. So this first entry can be removed from suffix array. Finally suffix array of T`

T` Suffix array = [3, 6, 0, 1, 2, 4, 5]

Usually, entry of sentinel number is not considered at the time of merging rather than removing after construction of suffix array.

Recursive call is completed and algorithm will resume back to R_{12} order, step 2.

Resuming at R_{12} Order

Suffix array of T` is R_{12} order but it cannot be used directly, it has to be remapped because sequences of three numbers were replaced by their R_{12} order when creating T`. For example element at index 0 in T` is actually R_{12} order of sequence of three numbers start at index 1 in T, similarly element at index 1 in T` represent order of sequence start at index 4 in T. As you can understand, index 0 in T` is mapped to index 1 in T, index 1 of T` is mapped with index 4 of T. Similarly index 3 of T` actually represent order of sequence start at index 10 in T, how you can calculate, there are three numbers before index 3 in T` and T have total four numbers having property $1 == \text{index} \% 3$ and first of such a index is index 1 of T, so $1 + (3 \times 3) = 10$.

What has explained about how to remap indexes of suffix array of T` to indexes of T is for explaining the logic behind it, but again the fastest way to remap is to use R_{12} indexes. R_{12} indexes are considered very carefully and intelligently, that's why it first includes indexes having property of $1 == \text{index} \% 3$ and then indexes having property $2 == \text{index} \% 3$. What we were calculating, R_{12} indexes is already written in similar way, let's see how R_{12} indexes will help to remap indexes.

R_{12} Indexes =	1	4	7	10	2	5	8
Index =>	0	1	2	3	4	5	6

T` Suffix array = [3, 6, 0, 1, 2, 4, 5]

Fig 12.0

First element of T` suffix array is 3 and due to that $R_{12}\text{Indexes}[3]$ will be inserted in R_{12} order

$R_{12} = [10,$

Then, second entry of T` suffix array is 6, so $R_{12}\text{Indexes}[6]$ that is 8, will be inserted

$R_{12} = [10, 8,$

Similar for all element of T` suffix array and the final R_{12} order would be,

$R_{12} = [10, 8, 1, 4, 7, 2, 5]$

R_{12} order is found, next step is to find R_0 order.

R_0 Order

Concept of find R_0 order is already explained in detail at the time of finding R'_0 order. R_0 indexes are highlighted by orange underline,

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	<u>0</u>	1	2	<u>3</u>	4	5	<u>6</u>	7	8	<u>9</u>	10	11	12

R_0 Indexes = [0, 3, 6, 9]

$R_0 = [97, 0], [97, 3], [97, 6], [98, 9]$

Quick reminder, in each pair of R_0 , first element is $T[\text{index}]$ and second element is index itself. Radix sort try to find the order of R_0 by comparing first element of each pair,

$R_0 = [\underline{97}, 0], [\underline{97}, 3], [\underline{97}, 6], [\underline{98}, 9]$

After first round of radix sort,

$R_0 = [\underline{97}, 0], [\underline{97}, 3], [\underline{97}, 6], [\underline{98}, 9]$

As you can see three pairs have same value and first round of radix sort is unable to find unique order. After first round of radix sort there are two buckets highlighted by different colors underline. Only first bucket has more than one pair. Second round of quick sort is required for each bucket having more than one pair, in this case only for first bucket. To perform second round, first element of each pair has to be modified

Usually when values are equal on indexes then we compare values on next indexes or in other words, if numbers are equal we compare next numbers. Values at indexes 0, 3 and 6 are equals then values at next indexes 1, 4, 7 should be compared but number wise comparison is not required because indexes 0, 3 and 6 are all belongs to R_0 indexes and after adding one they shall refer to R_{12} indexes, $1 == (0 + 1) \% 3$, $1 == (3 + 1) \% 3$ and $1 == (6 + 1) \% 3$. No number wise comparison required because R_{12} order is already known and by comparing R_{12} order pairs can be sorted.

First element of each pair will be replace by R_{12} order as follows,

$R_{12} =$	10	8	1	4	7	2	5
Index =>	0	1	2	3	4	5	6

Fig 13.0

First pair is [97, 0], second element is 0 and next index $0 + 1 = 1$ and if you see in fig 13.0 above index 1 in R_{12} is at position 2 and due to this first pair will be updated to $[2 + 1, 0]$. Similar for other pairs as follows,

[Position of 1 in R_{12} plus one, 0] \Rightarrow 1 is at pos = 2 \Rightarrow $[2 + 1, 0] \Rightarrow [3, 0]$

[Position of 4 in R_{12} plus one, 3] \Rightarrow 4 is at pos = 3 \Rightarrow $[3 + 1, 3] \Rightarrow [4, 3]$

[Position of 7 in R_{12} plus one, 6] \Rightarrow 7 is at pos = 4 \Rightarrow $[4 + 1, 6] \Rightarrow [5, 6]$

Last pair is not required to modify because it is in separate bucket and that bucket has no other elements which means its order is unique.

Second round of radix sort on first bucket,

[3, 0], [4, 3], [5, 6]

As you can see first element of all three pairs are different numbers and no number is equal to other number. After second round, unique order of R_0 is found as follows,

$R_0 = [3, 0], [4, 3], [5, 6], [98, 9]$

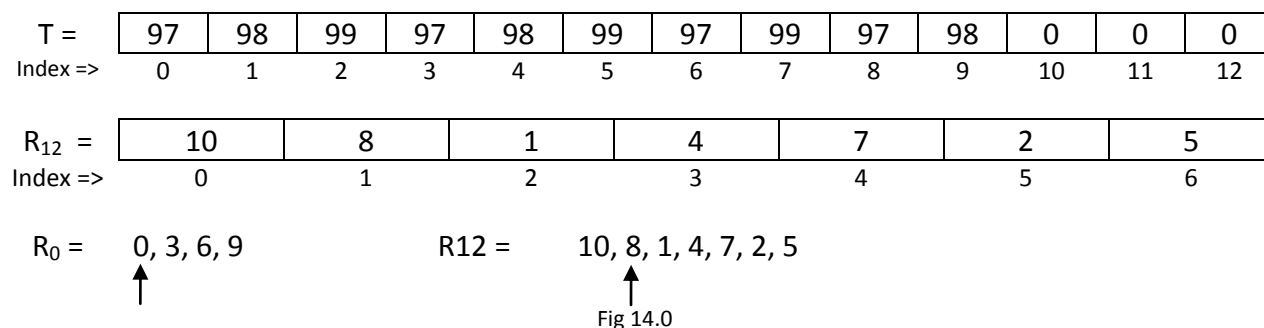
R_0 order is found and now R_0 is written in sorted order, we don't need to write these pairs rather these pairs shall be replaced by their respective index (second element). Finally R_0 order is

$R_0 = [0, 3, 6, 9]$

After R_0 order found last step is remain and that is merging R_0 and R_{12} order.

Merging R_0 and R_{12} orders

The concept of merging is explained at the time of merging R_0 and R_{12} orders. Here we can quickly start merging R_0 and R_{12} orders.



As mentioned previously that we don't have to include entry of sentinel number in suffix array and right way is to ignore it at the time of merging rather than removing it after suffix array is

T[0] value is 97 and T[8] is also 97, numbers are equal and 8 have property $2 \equiv 8 \% 3$. Next number T[0+1] and T[8+1] shall be compared. But again, T[1] and T[9] are equal, now next numbers are not required to be compared because $2 \equiv (0 + 2) \% 3$ and $2 \equiv (8 + 2) \% 3$ are both indexes are R_{12} indexes and R_{12} order is known. As you can see above in fig 14.0, In R_{12} , 10 occur at position zero and 2 is at position 5, It means suffix starts at 8 is smaller than suffix start at 0, so 8 will be merged.

$$R_0 = \begin{matrix} 0, 3, 6, 9 \\ \uparrow \end{matrix} \qquad R_{12} = \begin{matrix} 10, 8, 1, 4, 7, 2, 5 \\ \uparrow \end{matrix}$$
$$R_0 = 0, 3, 6, 9 \quad R_{12} = 10, 8, 1, 4, 7, 2, 5$$
$$R_0 = 0, 3, 6, 9 \quad R_{12} = 10, 8, 1, 4, 7, 2, 5$$

T =	97	98	99	97	98	99	97	99	97	98	0	0	0
Index =>	0	1	2	3	4	5	6	7	8	9	10	11	12

R ₁₂ =	10	8	1	4	7	2	5
Index =>	0	1	2	3	4	5	6

$$R_0 = 0, 3, 6, 9 \quad R_{12} = 10, 8, 1, 4, 7, 2, 5$$

Then, T[9] and T[1] shall be compared, but both T[9] and T[1] are 98. Numbers are equal but next numbers are not required to be compared because $1 == (9 + 1) \% 3$ and $2 == (1 + 1) \% 3$ both indexes belong to R_{12} indexes and R_{12} order is known. As you can see above in fig 15.0, In R_{12} , 10 is at pos zero and 2 is at pos 5. Order of 10 is smaller and due to this 9 will be merged,

Suffix array = [8, 0, 3, 6, 9]

$$R_0 = 0, 3, 6, 9 \quad R_{12} = 10, 8, 1, 4, 7, 2, 5$$

After merging 9, R_0 elements are exhausted and no elements remain. Elements remain in R_{12} will be appended in suffix array,

Suffix array = [8, 0, 3, 6, 9, 1, 4, 7, 2, 5]

Suffix array is constructed and all steps are completed.

Code

C++ implementation of DC3 algorithm is available under MIT license. Code also contains a main.cpp file to demo how to use the implementation with an example and print the result of suffix array and respective suffix in very illustrative way.

Code available under MIT license at: <https://github.com/vikasawadhiya/DC3-Algorithm>

14 August 2022, India