April 2025, India

Sparse Table

Tutorial

Tutorial By

Vikas Awadhiya

This work is licensed under CC-BY 4.0

This work is licensed under Creative Commons Attribution 4.0 International (CC BY 4.0). To view a copy of this license, visit

https://creativecommons.org/licenses/by/4.0/

Tutorial

Version 1.1

Tutorial By
Vikas Awadhiya
LinkedIn profile: https://in.linkedin.com/in/awadhiya-vikas

Introduction

If a sequence of 150 elements is being queried for a minimum value of inclusive range [10, 50], then the query can be answered in O(n) linear time complexity by simply comparing from index 10 up to index 50 and it is an efficient solution, but if the sequences is queried fifty thousand times for the minimum values of different ranges, then this is not an efficient solution.

Sparce Table is an auxiliary data-structure which requires O (1) time to answer the range queries. This auxiliary data-structure can be constructed in O (n log n) time using dynamic programming. So, the overall time complexity is one-time sparse table creation and constant time for each query.

Sparse table can be used in any problem where sequence is either constant or doesn't change after the sparse table creation for example LCP array of a suffix array never changes for a giver string and in few problems minimum LCP values are required to be find in different ranges of LCP array.

Sparse table contains index of minimum element of a range rather than storing the minimum element itself.

Let's consider a sequence of numbers [1, 4, 2, 0, 9, 7, 8, 3, 5, 6] and its sparse table as shown below in the fig 1.0.

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

Sparse Table =										
0	0	1	2	3	4	5	6	7	8	9
1	0	2	3	3	5	5	7	7	8	
2	3	3	3	3	7	7	7			
3	3	3	3							

Fig 1.0

To understand the sparse table, it is required to understand what each row represents in sparse table. Let's consider row_3 of sparse table as shown above in the fig 1.0.

Each entry of row_3 contains the index of minimum element of a range of length 8 and due to this complete row_3 contains the minimum element indices of all the possible ranges of length 8 of a sequence. Why the row_3 contains the minimum element information about the 8 elements long ranges of a sequence? So, the document discusses this in next section.

There are only three ranges of length 8 are possible in the sequence as highlighted by orange boundary below in the fig 2.0.

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

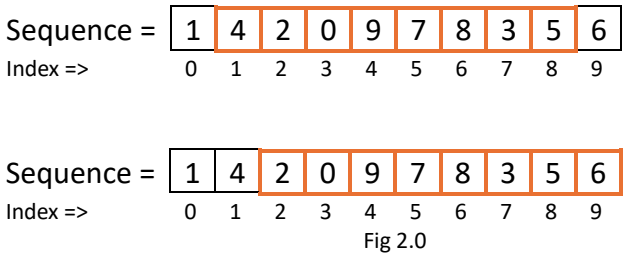


Fig 2.0

As shown above there are three possible ranges of length 8 and these ranges start from index 0, 1 and 2. The minimum element of a range starts at index 0 is at index 3 and the minimum element of ranges start at index 1 and index 2 is also at index 3 and due to this the row_3 contains index 3 in all of its entries as highlighted by orange boundary below in the fig 3.0.

Row =										
0	0	1	2	3	4	5	6	7	8	9
1	0	2	3	3	5	5	7	7	8	
2	3	3	3	3	7	7	7			
3	3	3	3							

Fig 3.0

The next section explains how the sparse table can be constructed in O (n log n) time complexity. It also explains the remaining aspects of the sparse table.

This document explains the Sparse Table for minimum element range query but sparse table can be used for other type of queries for example maximum element range query. Similarly, the document considers sequence of numbers or to say arrays of numbers to explain the concept but sequence could be the sequence of any type until elements of that type can be compared with each other.

Sparse Table Creation

Sparse table contains information about all possible ranges of a specific length of a sequence in a specific row. So, creation of the sparse table can be viewed as creation of individual rows.

Let's consider the previous sequence again as show below in the fig 4.0.

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

Fig 4.0

Row_0

This row represents information about the ranges of unit length. It helps to answer the queries of unit length ranges like range [1, 1]. It may feel redundant because a unit length range have only a single element and no comparison is required to find the minimum element. In the implement this row can be avoided but here it is included for explaining purpose. Row_0 contains all indices of a sequence because it represents unit length ranges, as shown below in fig 5.0.

Row = 0	0	1	2	3	4	5	6	7	8	9

Fig 5.0

To find what is the range length a row represents, can be found by the index of the row using following formula,

Range Length = $2^{\text{row index}}$

That's why row_0 contains information about ranges of unit length ($2^0 = 1$). So, the table[rowIndex][index] represent information about the range starts at ***index*** in the sequence and having length $\boldsymbol{2^{rowIndex}}$. Now, the only question remains is, why the range lengths are in the power of 2? So, the document discusses this in next section.

Row_1

This row represents information about the ranges of length 2 ($2^1 = 2$). Let's imagine a sliding window of two elements long, sliding on the sequence, as highlighted by blue boundary. It starts from index 0 and slides up to index 8 that's why row_1 contains 9 elements in sparse table. See the fig 6.0 below.

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

Fig 6.0

Here the minimum value in the range of length 2, starts at index 0, is element of value 1 at index 0 and that's why sparse table has index 0 as a value at table[1][0]. Here table[1][ ]

because range is of length 2 and table[ ][0] because range starts at index 0 in the sequence. It highlighted by orange boundaries in the fig 7.0 below.

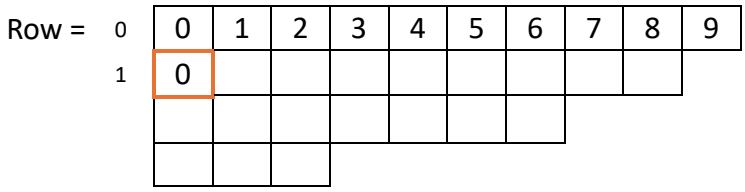
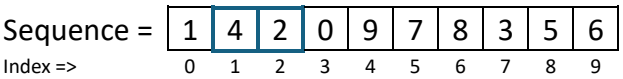


Fig 7.0

Window slides one position to the right and now range starts at index 1 as shown below in the fig 8.0.



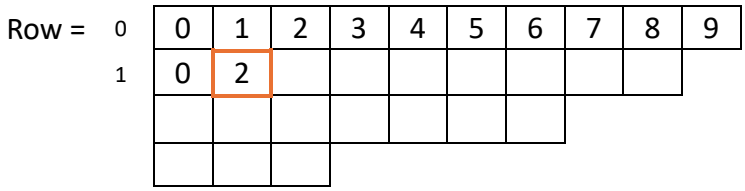


Fig 8.0

The minimum value in the range [1, 2] is element of value 2 at index 2 and due to this table[1][1] contains the index 2 as a value, as highlighted by orange boundary in the fig 8.0 above. Similarly other values of row_1 can be evaluated.

Row_2

This row represents information about ranges of length 4 ($2^2 = 4$). From row_2 onwards the dynamic programming comes into the picture. Let's imagine a sliding window of 4 elements long sliding on the sequence as shown below in the fig 9.0.

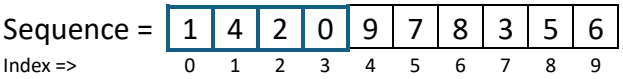


Fig 9.0

Now the element wise comparison is not required to find the minimum element in a range. This row represents the ranges of length 4 which is exactly double the length of the ranges represented by previous row. So, this range can be covered by two ranges of length 2 as shown below in the fig 10.0.

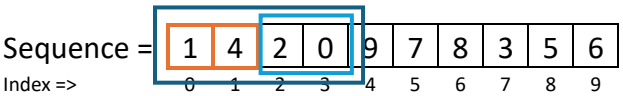


Fig10.0

As show above the minimum value of a range of length 4 can be found by comparing the minimum values of ranges of length 2 highlighted by orange boundary and the range highlighted by light blue boundary and this information is already available in sparse table at

table[1][0], range highlighted by orange boundary and table[1][2], range highlighted by light-blue boundary as shown below in the fig 11.0.

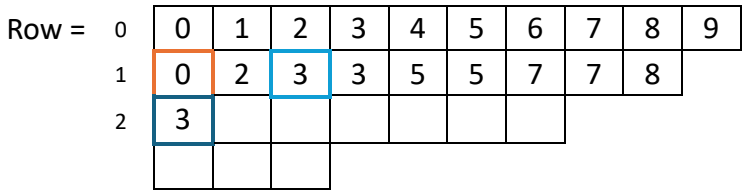


Fig 11.0

The sequence[table[1][0]] is 1 and sequence[table[1][2]] is 0 and due to this table[2][0] have index 3 (table[1][2]) as a value. It is visible that the sequence[0, 3] has minimum value 0 at index 3. The sliding window of length 4 can slide max up to index 6 on the sequence that's why row_2 have 7 elements. Similarly remaining values of row_2 can be evaluated.

The important point to observe here is, in sparse table construction, only one comparison is required to find the minimum value of any range. For a range of length 2 is it obvious and for range having length greater than 2 (2^p), also requires one comparison between the minimum values of two smaller sub-ranges.

Row_3

This row represents information about ranges having length 8 (2^3= 8). Let's imagine a sliding window of length 8 on the sequence as shown below in the fig 12.0.

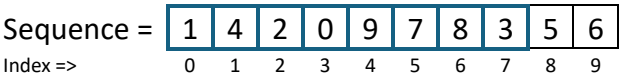


Fig 12.0

The minimum element index of this range can be found by comparing the two sub-ranges of length 4 as highlighted by orange and light blue boundaries below in the fig 13.0.

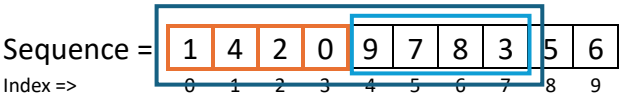


Fig 13.0

The minimum value of range[0, 7] can be found by comparing the minimum values of range [0, 3] (sequence[table[2][0]]) and rang[4, 7] (sequence[table[2][4]]) as show below in the fig 14.0. Similarly, the reaming values of row_3 can be found. Sliding window can slide to the right max up to index 2 on the sequence and that's why row_3 of sparse table has 3 elements.

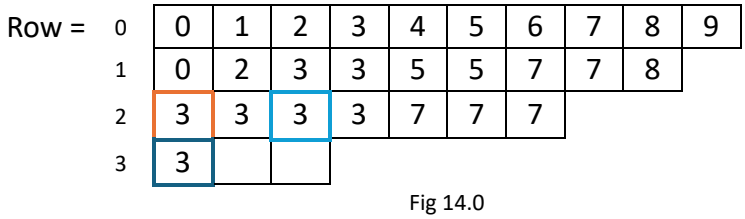


Fig 14.0

After evaluating the row_3 the sparse table creation ends because the row_4 represents the information about the ranges of length 2^4 = 16, but the sequence only has 10 elements.

Range query

Sparse table as an auxiliary data structure contains the minimum element information about various ranges of a sequence. These ranges of length 2^p can cover the ranges of any arbitrary length and due to this, queries can be answered in O(1) constant time complexity. Let's reconsider the previous sequence and sparse table created for it and see few random queries.

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

Row =											
	0	0	1	2	3	4	5	6	7	8	9
	1	0	2	3	3	5	5	7	7	8	
	2	3	3	3	3	7	7	7			
	3	3	3	3							

Fig 15.0

Range[0, 6]

The query range length is $7 \neq 2^p$. It means two sub-ranges are required to answer the query but sparse table has many ranges of different lengths (2^p), so it is first required to find what should be the length of sub-ranges which can cover this range that is ultimately to find the value of p for sub-ranges.

Query Range Length = (upper bound index – lower bound index) + 1

The range is inclusive and includes both lower and upper bound indices.

So, the query range length = (6 – 0) + 1 = 7, and p can be found as follows,

$P = \lfloor \log_2(\text{query range length}) \rfloor = \text{floor}(\log_2(\text{query range length}))$

Here query range length is 7 and the $p = \lfloor \log_2(7) \rfloor = \lfloor 2.8074 \rfloor = 2$. So, the sub-ranges of length 4 (2^2) shall be used to answer the query and the first sub range starts at index 0, because query range starts at index 0. The sub-range highlighted by orange boundary in the fig 16.0 below.

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

Fig 16.0

But three elements are remained to be covered by sub-range, so imagine a range of length 4 ends at index 6 in the sequence. A range of length 4 ends at index 6 should start at index = (6 – 4) + 1 = 3 as highlighted in light blue boundary in the fig 17.0 below.

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

Fig 17.0

So, the minimum value of range[0, 6] can be found by a single comparison between the minimum values of range[0, 3] as range[3, 6] as follows,

Range[0, 6] minimum value = min(sequence[table[2][0]], sequence[table[2][3]]) = 0

Here table[2][] is used because row_2 contains information about the ranges of length 4.

Range[1, 5]

Query range length = (5 – 1) + 1 = 5 ≠ 2^p,

P = ⌊ $\log_2(5)$ ⌋ = ⌊2.3219⌋ = 2,

Sub-range length = $2^p = 2^2 = 4$,

Range[1, 5] minimum value = min(sequence[table[2][1]], sequence[table[2][2]]) = 0

As shown below in the fig 18.0.

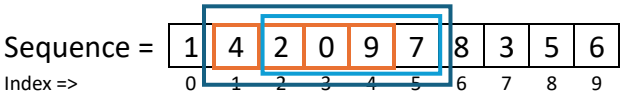


Fig 18.0

Range[0, 9]

This query covers entire sequence or in other words finding the minimum in the sequence.

Query range length = (9 – 0) + 1 = 10 ≠ 2^p,

P = ⌊ $\log_2(10)$ ⌋ = ⌊3.3219⌋ = 3

Sub-range length = $2^p = 2^3 = 8$

Range[0, 9] minimum value = min(sequence[table[3][0]], sequence[table[2][2]]) = 0

As shown below in the fig 19.0.

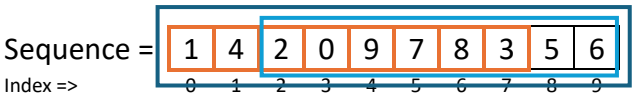


Fig 19.0

Range[4, 7]

Query range length = (7 – 4) + 1 = 4 = $2^2 = 2^p$,

It is a range of length 2^p. Information about the minimum value of this range is directly available in the sparse table. The minimum value of the range[4, 7] is sequence[table[2][4]] = 3, as shown below in fig 20.0

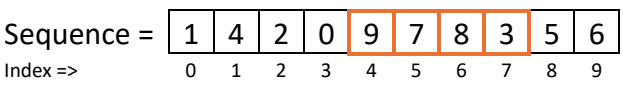


Fig 20.0

Similarly, other queries can be answered in constant time complexity.

Implementation

The C++ implementation of the Sparse Table is provided under the MIT License. It also contains the “main.cpp” file to demonstrate its usage.

Code available under MIT License at: https://github.com/vikasawadhiya/Sparse-Table

The implementation returns index of minimum element as an answer of range minimum query rather than element itself.

This implementation can be used with any type of sequence if the less than operator is defined for that type and two objects of that type can be compared using < operator. If you want to use sparse table for other types of queries like range maximum query then you can implement a more generic sparse table.

April 2025, India