# Sparse Table

Sparse table can answers range minimum query in constant time of non changing sequence. Sparse table for non modifying sequence can be created by dynamic programming in O(nlog n) time and O(nlog n) space complexity. Once sparse table created for a sequence, range minimum queries can be answered in O(1) time complexity. Sparse table can help other algorithms in case like finding minimum value between two values of longest common prefix array (LCP array). LCP array is an auxiliary data structure of Suffix array.

**VIKAS AWADHIYA**

**LinkedIn profile: https://in.linkedin.com/in/awadhiya-vikas**

# Introduction

If a non changing sequence of 150 elements is being queried for minmum value of inclusive range [10, 50], then it can be answered in O(n) time by simply comparing from index 10 up to 50 and it is efficient solution, but now image if a same sequences is queried fifty thousand times for minimum value of different ranges then this solution is not best solution and the same can be done in more efficiently.

Using sparse table a minimum query of a range of a non changing sequence can be done in O(1) time. But to answer query in constant time requires sparse table to be created. Dynamic programming approach can create sparse table in O(nlog n) time, so overall time complexity is one time table creation and then constant for each query.

Sparse table can used in any application where sequence is none changing like lcp array of a suffix array never changes for a giver string and in few problems minimum lcp value between two lcp values is need to be queried.

Sparse table contains minimum range value information in terms of index rather than storing values directly. For example sparse table for sequence [1, 4, 2, 0, 9, 7, 8, 3, 5, 6] is showed below in fig 1,

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sequence

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 | | | | | | | |

Sparse table

Fig 1

First of all what each row and its columns represent in sparse table? To understand what each row represents, let's take row 3 as example and see what it represents,

Row 3 contains all ranges of length 8 possible in sequence, why range of length 8 for row 3? It will be clear in table creation process but for now understand that each row represent all possible ranges of specific length in given sequence,

There are only three ranges of length 8 possible in sequence showed as orange boundary highlighted, below,

| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Fig 2

Fig 2 show first range start from index = 0,

| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Fig 3

Fig 3 represents range of length 8 starts from index = 1,

| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Fig 4

Fig4 represents last range of length 8 starts from index = 2 of sequence. There are all ranges of length 8 possible in given sequence and that's why row 3 contains minimum value indexes of these three ranges in row 3. Index of minimum value of range start at index = 0 of sequence will save at index zero of row 3 means table[3][0], range start at index = 1 will go for index 1 of row 3, it means sparse table keep index of minimum value of a range in row respectively to its start index.

After introduction, next step is to understand how dynamic programming helps to create sparse table in complexity of  O(nlog n) time.

## Sparse table creation

To understand how sparse table is being created, it has to be views in terms of individual rows, what a row represent in table and how the entries of a row shell be filed, is way to understand sparse table and its creation process.

To understand what each row represents let's take previous sequence again as showed in fig5 below,

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Fig 5

### Row 0

Let's consider a query is raised to find the minimum value of range [0, 0], it means a range of only one element and in such case noting is required to perform, because there is no other element to compare. Each element of sequence is a range of length 1 which starts and end on same index and due to this row 0 contains all index of sequence, as highlighted in orange color and showed in fig 6,

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sequence

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 3 | 5 | 5 | 7 | 7 | 8 | |
| 3 | 3 | 3 | 3 | 7 | 7 | 7 | | | |
| 3 | 3 | 3 | | | | | | | |

Sparse table

Fig 6

Sparse table row 0 contains all ranges of length 1 of sequence, but why it is needed? A single element range is redundant range for query. The answer is, in implementation row 0 can be avoided but it is included to explain dynamic programming approach.

Each row contains index of minimum value information of all possible ranges of specific length and specific length is directly related to index of row in sparse table, in simple word a row represents all ranges of **length = $2^{\text{row index}}$**, that's why row 0 has information of range length = $2^0$ = 1.

A question raises here why ranges are in power of 2? It can be answered in finding minimum section of this document and until that please wait!

## Row 1

Range length will be $2^1 = 2$, this time imagine a window of length 2 sliding on sequence as highlighted in purple color boundary in fig 7 below,

| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sequence

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 3 | 5 | 5 | 7 | 7 | 8 | |
| 3 | 3 | 3 | 3 | 7 | 7 | 7 | | | |
| 3 | 3 | 3 | | | | | | | |

Sparse table

Fig 7

As showed above in fig7, a sliding window represents current range of length 2, and this range starts from index 0 of sequence. So in this current range, minimum value between 1 and 4 is 1 and range starts from index = 0 that's why index of minimum value will be copied in zero position of row1 means table[1][0] as highlighted in orange color filled cell in row 1 of sparse table above in fig 7.

After it window slides by one position as showed below in fig 8,

| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sequence

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 3 | 5 | 5 | 7 | 7 | 8 | |
| 3 | 3 | 3 | 3 | 7 | 7 | 7 | | | |
| 3 | 3 | 3 | | | | | | | |

Sparse table

Fig 8

As showed above in fig 8, minimum value between 4 and 2 is 2 and start index = 1, that's why index of minimum value will be copied at table[1][1] as highlighted orange color cell in row 1 in fig8 above.

As showed above for first two ranges of length 2, same can be done for all ranges. Sliding window can slider up to max index = 8 in subsequence that's why in sparse table row 1 has one column less than row 0.

## Row 2

From this row, dynamic programming will be in action. At row 2, range length will be $2^2 = 4$, so imagine a sliding window of length 4, sliding on sequence, highlighted in purple color boundary as showed below in fig 9,

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Fig 9

But now finding minimum value of range by comparing all elements is not required, because length of boundary is exactly double of boundary length of previous row. It means this range can be covered by combining two small ranges as showed in fig10 below,

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Fig 10

As showed above minimum value of range length 4 can be found by comparing minimum value of orange boundary range and light blue boundary range. As showed in fig10, orange boundary range starts at index = 0 and light blue boundary range starts at index = 2, and minimum value indexes of these boundary would be table[1][0] and table[1][2] respectively.

Finding minimum value of range can be done in constant time by comparing minimum value of small ranges sequence[table[1][0]] and sequence[table[1][2]], and sparse table will updated as showed in orange cell of row2 in fig11 below,

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Value | 1 | 4 | 2 | 0 | 9 | 7 | 8 | 3 | 5 | 6 |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sequence

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 2 | 3 | 3 | 5 | 5 | 7 | 7 | 8 | |
| 3 | 3 | 3 | 3 | 7 | 7 | 7 | | | |
| 3 | 3 | 3 | | | | | | | |

Sparse table

Fig 11

As demoed above for one range of length 4, same can be done for other ranges. Current sliding window is of length 4 and it can slide max up to index = 6 in sequence and due to this row 2 have two columns less then row 1.

### Row 3

At row 3 range length will be $2^3 = 8$, here again imagine a sliding window of length 8, sliding on sequence as showed below fig12,

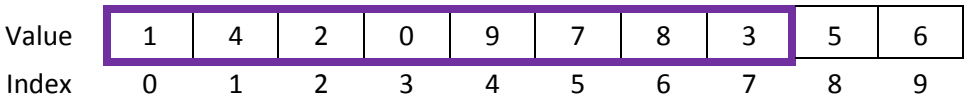


Fig 12

Minimum value of range required to find by comparing all elements is only required for range of length 2 and after it all ranges of length above 2 can be calculated by comparing sub ranges minimum values in constant time. In this case range of length 8 can be covered by combining two sub ranges of length 4 as showed below in fig 13,

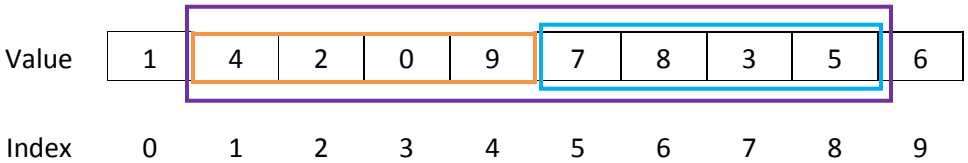


Fig 13

Minimum value of all range of length 8 can be found by comparing sub ranges minimum values in constant time. Sliding window can sliding only three times in sequence that's why row 3 has only 3 elements.

Sparse table creation process for this sequence will completed with completion of row 3 because after row 3, range length will become $2^4 = 16$, but sequence has only 10 elements.

Next point to understand is how sparse table find minimum value of a random range in constant time?

## Finding Minimum

As explained in table creation process, rows of table save index of minimum value of different range length, starting from all possible indexes. Question raised their, why range length is in power of 2? Answer is, by combining ranges of length are in power 2; any range of random length can be covered and this topic will demo it.

Sparse table is created for sequence [1, 4, 2, 0, 9, 7, 8, 3, 5, 6], let's take few random ranges and see how to find minimum values of these ranges,

**Range [0, 6]**: To answer range minimum query, need to find what is the length of range, which can be calculated as follows,

**Range length = (last Index – first Index) + 1**

As per above formula rang length = (6 – 0) + 1 = 7, next thing is find what would be the length of sub ranges that shall cover this range and that can be calculated as follows,

**P** = $\log_2$ **(range length)** (floor value)

**Sub range length =** $2^P$

As per above formula P = $\log_2$ 7 = 2 (floor value), and sub range length = $2^2$ = 4, It means sub ranges of length 4 can cover this original range as showed below,

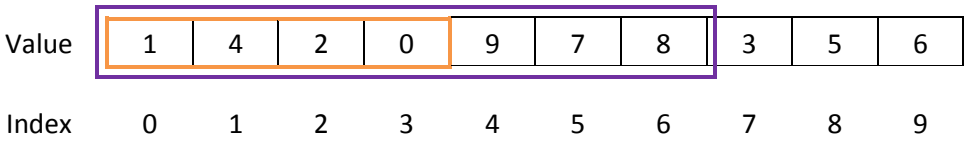


Fig 11

Range [0, 6] starts from index = 0, so consider sub range of length 4 also start from index = 0 as showed in fig 11 above in orange boundary. But orange boundary sub range cannot cover range [0, 6] and three elements are left, and to cover these three elements think about a sub range of length 4, which should end at index = 6 and that sub range will start from index = 3, as highlighted in light blue boundary as showed in fig 12 below,

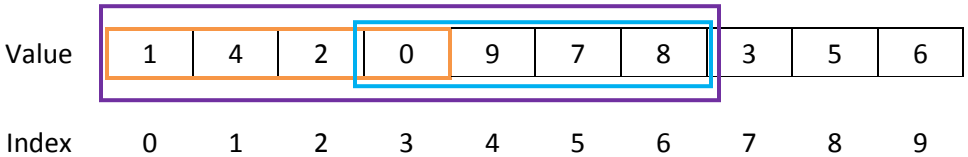


Fig 12

As showed above in fig12, to find range [0, 6] minimum value, only requires to compare minimum values of range [0, 3] and range [3, 6] and that can be done in constant time,

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

Let's consider two more examples,

**Range [1,5]**: As explained in previous example,

Rang length = (5 – 1) + 1 = 5,

P = $\log_2 5$ = 2 (floor value)

Sub range length = $2^2$ = 4 and,

Rang [1, 5] minimum value = min (sequence[table [2][1]], sequence[table[2][2]]) as showed in fig 12 below,

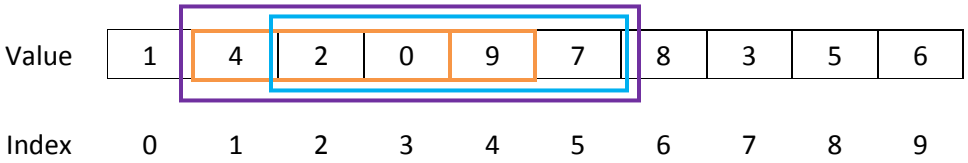


Fig 12

**Range [0,9]**: Complete sequence then,

Rang length = (9 – 0) + 1 = 10,

P = $\log_2 10$ = 3 (floor value)

Sub range length = $2^3$ = 8 and,

Rang [1, 5] minimum value = min (sequence[table [3][0]], sequence[table[3][2]]) as showed in fig 13 below,

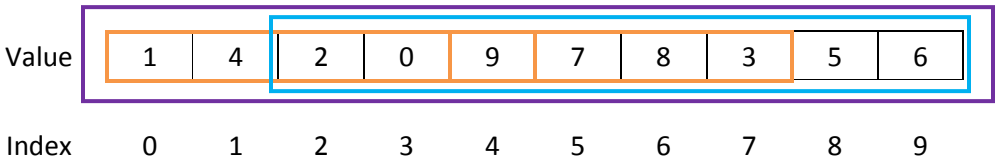


Fig 13

## C++ Code

C++ code is available under MIT licenses at: https://github.com/vikasawadhiya/Sparse-Table

Here code is included only for completion of document.

sparsetable.hpp

```
/*******************************************************************************
** MIT License
**
** Copyright (c) 2021 VIKAS AWADHIYA
**
** Permission is hereby granted, free of charge, to any person obtaining a copy
** of this software and associated documentation files (the "Software"), to deal
** in the Software without restriction, including without limitation the rights
** to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
** copies of the Software, and to permit persons to whom the Software is
** furnished to do so, subject to the following conditions:

** The above copyright notice and this permission notice shall be included in all
** copies or substantial portions of the Software.

** THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
** IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
** FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
** AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
** LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
** OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
** SOFTWARE.
*******************************************************************************/

#ifndef SPARSETABLE_HPP
#define SPARSETABLE_HPP

#include <cmath>
#include <vector>

template <typename T>
class SparseTable
{
public:
    SparseTable(const std::vector<T>& elements);

    std::size_t indexOfMinimumValue(std::size_t inclusiveFirstIndex,
std::size_t inclusiveLastIndex,
                    const std::vector<T>& elements) const;

private:
    std::vector<std::vector<T>> table;
};

template <typename T>
SparseTable<T>::SparseTable(const std::vector<T>& elements):
table(std::log2(elements.size())){

    table.front().reserve((elements.size() - 2) + 1);
```

```
    for(std::size_t index = 0, lastIndex = elements.size() - 2; index <=
lastIndex; ++index){

        table.front().push_back((elements[index] < elements[index + 1]) ?
index : index + 1);
    }

    std::size_t tableIndex = 1;

    for(std::size_t rangeLen = 4; rangeLen < elements.size(); rangeLen *=
2, ++tableIndex){

        std::size_t preTableIndex = tableIndex - 1;
        std::size_t preRangeLen = rangeLen / 2;

        table[tableIndex].reserve((elements.size() - rangeLen) + 1);

        for(std::size_t index = 0, lastIndex = elements.size() - rangeLen;
index <= lastIndex; ++index){

            table[tableIndex].push_back(
                        (elements[table[preTableIndex][index]] <
elements[table[preTableIndex][index + preRangeLen]])?
                        table[preTableIndex][index] :
table[preTableIndex][index + preRangeLen]);
        }
    }
}

template <typename T>
std::size_t SparseTable<T>::indexOfMinimumValue(std::size_t
inclusiveFirstIndex, std::size_t inclusiveLastIndex,
                                                const std::vector<T>
&elements) const{

    if(inclusiveLastIndex == inclusiveFirstIndex){

        return inclusiveFirstIndex;
    }

    std::size_t numOfElements = (inclusiveLastIndex - inclusiveFirstIndex)
+ 1;

    std::size_t tableIndex = std::log2(numOfElements) - 1;
    std::size_t rangeLen = 1 << (tableIndex + 1);


    if(numOfElements == rangeLen){

        return table[tableIndex][inclusiveFirstIndex];
    }
    else{
```

```
        std::size_t inclusiveNextIndex = inclusiveFirstIndex +
(numOfElements - rangeLen);

        if(elements[table[tableIndex][inclusiveFirstIndex]] <
elements[table[tableIndex][inclusiveNextIndex]]){

            return table[tableIndex][inclusiveFirstIndex];
        }
        else{
            return table[tableIndex][inclusiveNextIndex];
        }
    }
}


#endif // SPARSETABLE_HPP
```

## Summary

Sparse table can be created in O(nlog n) time complexity and it has O(nlog n) space complexity, once sparse table is created for non modifying sequence, minimum value between two elements can be found in O(1) time complexity.