Acta of Turin Polytechnic University in Tashkent, 2023, 30, 34-37
Published Online September 2023 in Acta TTPU (http://www.acta.polito.uz/)

DATA STRUCTURE SPARSE TABLE

Iskandarov I.Z.
Urgench branch of Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi
Email: islom.iskandarov@ubtuit.uz

Abstract– This article discusses the sparse table data structure: description, capabilities, and scope of application. The results of comparison with such data structures as a segment tree and a sqrt decomposition for two problems are shown. The listing codes are presented in the C++ programming language.

Key words– data structures, algorithms, sparse table.

I INTRODUCTION

Data structure is a way to store and organize data in order to facilitate access and modifications [1]. Each data structure has its own scope, that is, the types of tasks that it can solve. In this article, we will consider a sparse table data structure that stores data in a special (sparse) form and allows you to quickly respond to queries in a segment without modifying elements.

II THE METHODOLOGY

This data structure was introduced in the article The LCA Problem Revisited [2] as part of an optimized solution for the problem. The LCA problem is as follows: Let a tree G be given. Requests of the form (V1, V2) are received as input, and for each request it is required to find their least common ancestor, i.e., vertex V, which lies on the path from the root to V1, on the path from the root to V2, and from all such vertices, the lowest one should be chosen.

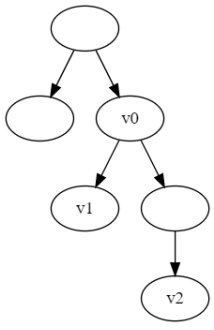

Fig. 1: Graph example for LCA

In the example shown in Figure 1 node v0 is LCA for nodes v1 and v2. The solution to the problem presented in this article is related to the RMQ (Range Minimum Query) problem of finding the minimum on a segment, and the sparse table data structure allows you to effectively solve this problem.

In the book Competitive Programmer's Handbook [3] in *Static array queries* section also shows a solution for querying the minimums with an example with a static array. This algorithm is also presented in Internet resources dedicated to data structures and algorithms. For example, in [4] this algorithm is presented with an implementation in the C++ language, the topic is considered in more detail in [5], including its application for commutative operations.

Formulation of the problem. Suppose we have an array of numbers A consisting of N elements and we need to find a certain value for some segment of the array A. Values can be, for example: the minimum element on the segment, the maximum element on the segment, the sum, etc. A sparse table allows for time and memory overhead $O(NlogN)$ for idempotent operations (minimum, maximum, GCD, etc.) respond to the request in time $O(1)$. Next, consider, as an example, the use of a sparse table to find the minimum element on a segment in queries.

Structure Description. The idea behind the sparse table data structure is the use of a table T for storing answers for segments. But we cannot store values for all segments, since the memory consumption will be $O(N^2)$, and we will store values for segments of length 2^k where $0 \leq k \leq [log_2 n]$. Formally in $T[l][j]$ the value for the segment with the left border l of length 2^j, that is, the value for the segment with indices $[l; l+2^j-1]$. For example, for an array with elements [3, 2, 4, 5, 1, 1, 5, 3] of length 8 the table would look like this:

rmq[3]:	1						
rmq[2]:	2	1	1	1	1		
rmq[1]:	2	2	4	1	1	1	3
rmq[0]:	3	2	4	5	1	1	5	3

Fig. 2: Table structure for 8 elements

Building. Consider the construction for storing the value

of the minimum element on the segment. First of all, we need to calculate the values of binary logarithms; we will calculate them as integers rounded down. Let's do a pre-calculation and store the values in an array:

```
1. int lg[N+1];
2. lg[1]=0;
3. for (int i=2; i<=N; ++i){
4.     lg[i]=lg[i/2]+1;
5. }
```

Listing 1. Calculation of values of binary logarithms of numbers.

The next step is to build the table itself:

```
1. for (int i=1; (1<<i) <=N; ++i) {
2.     int len = (1<<i);
3.     for (int j=1; j+len-1 <=N; j++) {
4.         int tail = j+len-1;
5.         T[j][i]=min(T[j][i-1], T[tail-len/2+1][i-1]);
6.     }
7. }
```

Listing 2. Building a sparse table for the minimum

Getting answers. To obtain a value for the segment $[l;r]$ we "combine" the answers for the two segments $[l;l+2^j-1]$ and $[r-2^j+1;r]$, where j the largest integer for which $2^j \leq r-l+1$, i.e. $j=[log_2(r-l+1)]$.

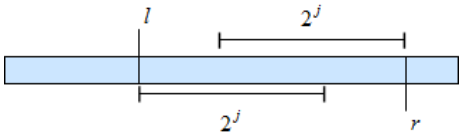

Fig. 3: Combining segments to answer a query

```
1. int get (int l, int r) {
2.     int len = r-l+1;
3.     int p=lg[len];
4.
5.     return min(T[l][p], T[r-(1<<p) + 1][p]);
6. }
```

Listing 3. Receiving a response to a request for a minimum in a segment

Further in requests we can use this function:

```
1. int Q;
2. cin >> Q;
3. for (int i=1; i<=Q; i++) {
4.     int l;
5.     int r;
6.     cin >> l >> r;
7.     cout << get(l, r) << "\n";
8. }
```

Listing 4. Getting an answer to a query

The above example can be used for other idempotent operations, for example, for finding the maximum, GCD, etc. by changing the calculation function for building a table and receiving an answer.

III RESULTS AND DISCUSSION

Here is a comparative analysis of a sparse table and other data structures for various tasks (queries) and data size. As a comparison metric, we take the execution time of programs using these data structures in the C++ language. The calculation includes the total time spent (in milliseconds) on the following operations: building a structure, reading requests from a file, calculating and outputting results to a file[1]. The size of the array is denoted by N, the number of requests is equal to Q.

Structure / Data size	$N=10^4$ $Q=10^4$	$N=10^5$ $Q=10^5$	$N=10^5$ $Q=10^6$	$N=10^6$ $Q=5*10^6$
Segment tree	56 ms	619 ms	5754 ms	34178 ms
SQRT-decomposition	63 ms	806 ms	7235 ms	58415 ms
Sparse table	67 ms	802 ms	5594 ms	30761 ms

TABLE 1: COMPARISON OF STRUCTURES FOR THE PROBLEM RMQ

[1]Standard input operator cin and standard output operator cout were used to read and write data.

Structure / Data size	$N = 10^4$ $Q = 10^4$	$N = 10^5$ $Q = 10^5$	$N = 10^5$ $Q = 10^6$	$N = 10^6$ $Q = 5 * 10^6$
Segment tree	57 ms	613 ms	6387 ms	32323 ms
SQRT-decomposition	56 ms	762 ms	7058 ms	58890 ms
Sparse table	63 ms	722 ms	5586 ms	31661 ms

TABLE 2: COMPARISON OF STRUCTURES FOR THE PROBLEM OF FINDING GCD ON A SEGMENT

Structure / Data size	$N = 10^4$ $Q = 10^4$	$N = 10^5$ $Q = 10^5$	$N = 10^5$ $Q = 10^6$	$N = 10^6$ $Q = 5 * 10^6$
Segment tree	58 ms	634 ms	6863 ms	40786 ms
SQRT-decomposition	56 ms	890 ms	8863 ms	86058 ms
Sparse table	123 ms	722 ms	5922 ms	33616 ms

TABLE 3: COMPARISON OF STRUCTURES FOR THE PROBLEM OF FINDING SUM ON THE SEGMENT

Below are data histograms for problems RMQ and sum (Fig.4, Fig.5). Data sizes are divided into 4 groups: $A(N = 10^4, Q = 10^4), B(N = 10^5, Q = 10^5), C(N = 10^6, Q = 10^6), D(N = 10^6, Q = 5 * 10^6)$. Time values are calculated using the natural logarithm for a smoother presentation. More formally, the time value is replaced by the following expression:

$$Time = lnTime$$

Based on the data above, we can see that a sparse table gives a gain with a larger number of queries relative to the size of the input data. The first two tables refer to problems for which this algorithm calculates the answer in constant time. Additionally, an associative operation (sum) is given in the third table. For such operations, the structure calculates the response in logarithmic time. It is noteworthy that in this case the sparse table gives good results with large data sizes, especially in the latter case it gives a significant advantage.

Fig. 4: Comparison of structures for the problem RMQ

Fig. 5: Comparison of structures for the problem of finding sum on the segment

IV CONCLUSION

The sparse table data structure is appropriate to use in the following cases:

1. There is no element modification, that is, the values of the array elements do not change;
2. The number of queries is large;
3. It is necessary to find the value of an idempotent function for a segment such as minimum, maximum, etc. For such operations, a sparse table gives answers in time $O(1)$, in other cases for $O(logN)$ which no longer gives advantages over other data structures.

V REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, “Introduction to Algorithms”, 3rd ed, pp. 9.

[2] Antti Laaksonen, Competitive Programmer’s Handbook, Draft August 19, 2019. http://www2.compute.dtu.dk/courses/02282/2021/nca/CPbook.pdf

[3] M.A. Bender, M.Farach-Colton. The LCA problem revisited. In Latin American Symposium on Theoretical Informatics, 88–94, 2000.