CSC11F: Advanced Data Structures and Algorithms

Segment Tree

Yutaka Watanobe (142-A, yutaka@u-aizu.ac.jp)

1 Introduction

Answering for queries to extract or update data is a fundamental operation to a data collection. As the terminology 'query' is used for database management system, it includes search operation (e.g. find the minimum element) and modifications (e.g. update a specified element).

There are some possible queries for elements or intervals in an array as follows.

Queries to find the specified value

- **RMQ: Range Minimum Query.** Find the minimum/maximum value in the specified range.
- **RSQ: Range Sum Query.** Get the sum of values in the specified range.
- **SRQ: Single Read Query.** Find the element at the specified position i.
- etc.

Queries to modify the specified elements

- **RUQ: Range Update Query.** Update the values in the specified range to x.
- **RAQ: Range Add Query.** Add x to elements in the specified range.
- **SUQ: Single Update Query.** Update the value of the specified element i to x.
- etc.

It is easy to design and implement naive algorithms for such queries. For example, we can design $O(1)$ algorithms for reading/updating a single element, and $O(N)$ algorithms for calculating/updating values of elements in the specified range where N is the number of elements.

On the other hand, if you should deal with many such queries (where Q is the number of queries), you should consider more efficient algorithms. In other words, the algorithm should not be $O(QN)$. Fortunately, we can implement efficient algorithms by means of the advanced data structure called segment trees.

2 Structure of Segment Tree

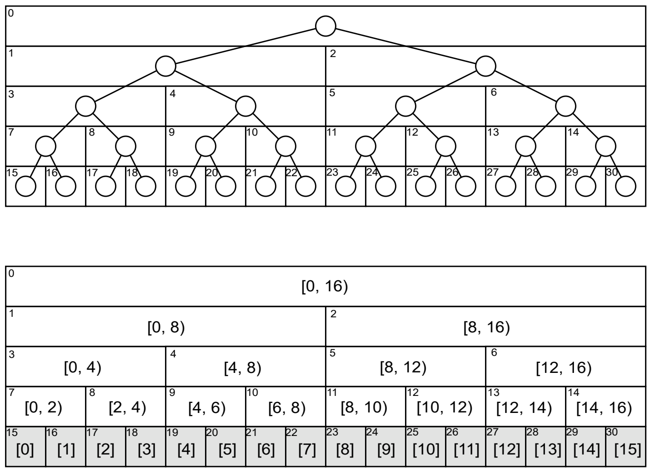

Figure 1: A structure of a segment tree

A segment tree T is a complete binary tree which each node represents an interval on an array as shown in Figure 1. Each node of the complete binary tree is indexed as follows:

- Index of the root is 0.
- Indices of the left child and the right child of node k is $2k+1$ and $2k+2$ respectively.
- Index of the parent of node k is $(k-1)/2$.

Each node of the segment tree T is to represent a single element or a sequence of elements in an interval of the array as follows:

- A 0-origin array A of size N is managed by a segment tree T of size $2N-1$.
- To employ a segment tree, we suppose the length of the array is a power of 2.
- Each leaf of the segment tree T corresponds to an element i of the array such that $0 \leq i < N$.
- An internal node of the segment tree T represents an interval $[i, j)$ where $0 \leq i < j \leq N$.

The root of the segment tree T represents the whole range $[0, N)$ of the array. The range $[0, N)$ represented by the root is broken down into two half segments $[0, \frac{0+N}{2})$ and $[\frac{0+N}{2}, N)$ at its children respectively. Generally, the range $[l, r)$ represented by a node is broken down into the half segments $[l, \frac{l+r}{2})$ and $[\frac{l+r}{2}, r)$ at its children respectively. The index of node in the segment tree for i-th element of the array is $i + N - 1$.

For example, the segment tree presented in Figure 1 is to manage an array with 16 elements where the node 15 corresponds to the first element of the array. The node 9 represents the value for the range [4, 6), and the node 5 represents the value for the range [8, 12), and so on.

3 RMQ and SUQ

Let's consider a problem to answer for **both** RMQ and SUQ for a sequence $A = \{a_0, a_1, ..., a_{n-1}\}$ with the following operations:

- find(s, t): report the minimum element in $a_s, a_{s+1}, ..., a_t$.
- update(i, x): change a_i to x.

Now we should answer for RMQ, so each node of the segment tree should maintain the minimum value between the corresponding interval.

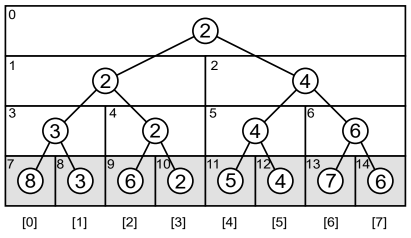

Figure 2: An example of segment trees for RMQ

Figure 2 shows an example of segment trees for RMQ. The value of the root is 2 which is the minimum value in the whole array, and for each sub tree, its root maintains the minimum value for the corresponding interval.

For example, the value of node 2 and node 5 is 4, because they include the minimum element 4 in their range [4, 8) and [4, 6) respectively.

Operation for SUQ

Managing the RMQ condition, let's update an element of the array. Figure 3 and 4 show two examples of the update operations where update(i, x) denotes "replace $A[i]$ with x".

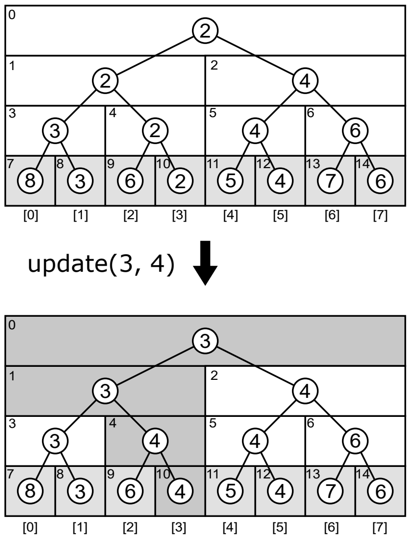

Figure 3: An example of update queries (1)

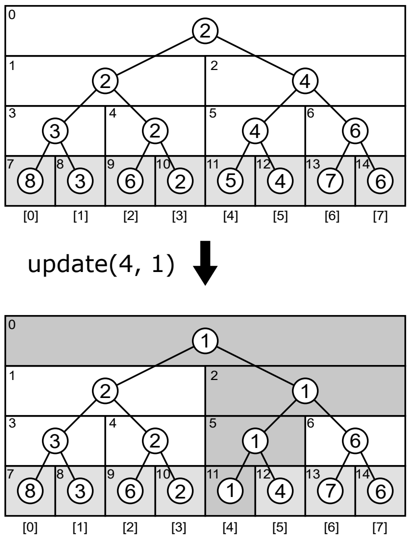

Figure 4: An example of update queries (2)

The update operation starts with the modification of the array element (a leaf of the segment tree) and it goes up the segment tree. When going up the tree, at node with index k, we take pairs of nodes with indices $(2k + 1, 2k + 2)$ and combine their values (for RMQ, select the minimum element). In this way, we can perform the single update query with at most $\log N$ (the height of the tree) steps where N is the number of nodes in the segment tree.

Operation RMQ

For a range query, we do not need to traverse all elements in the specified interval on the array, but scan narrow downed nodes of the segment tree.

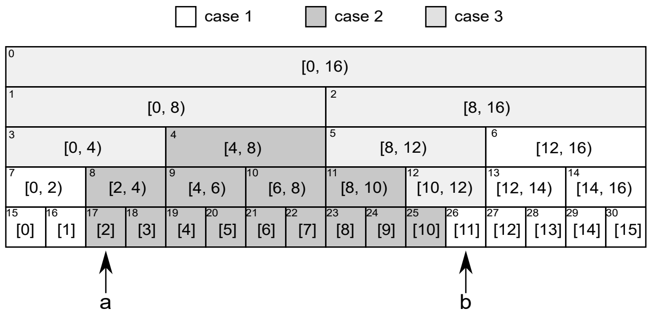

Figure 5: Traversal area for a range query

Figure 5 shows an example of the traversal area for a range query. The range query finds the solution by the post-order tree traversal starting with the root. Suppose the range of the query is $[a, b)$ and the range of the current node during the traversal is $[l, r)$. When we visit a node during the traversal, we need to consider the following cases:

1. the two ranges $[a, b)$ and $[l, r)$ do not intersect, say $r \leq a$ or $b \leq l$
2. $[a, b)$ includes $[l, r)$, say $a \leq l$ and $r \leq b$
3. others

For case 1, we should just return a value which does not affect the solution for the query. For RMQ, we should return INF (e.g. a large integer).

For case 2, we have the solution for this range, so we can immediately return the value of the node.

For case 3, we can not decide the solution in the range because there are possibilities to have the solution in both children of the current node. So, we should visit the children recursively to obtain the solutions, then select and return the optimal solution for the range. For RMQ, we should select the minimum value.

In this way, we can perform a range query in $O(\log N)$ where N is the number of nodes in the segment tree.

4 Implementation

For implementing a segment tree for RMQ, we can declare the following variables:

```
n:    the size of the target array
D[i]: minimum values for the range represented by
      node i of the segment tree
```

Procedures for the initialization, update and range queries for RMQ can be implemented as follows:

```
initRMQ(n_) // input the required array size
    n = 1
    // the size of array is power of 2
    while n < n_:
        n *= 2
    // initialize all node in the segment tree
    for i = 0 to 2*n-1-1:
        D[i] = INF


update(k, x)
    k += n - 1  // convert array index to tree index
    D[k] = x
    while  k > 0:
        k = (k - 1)/2
        D[k] = min(D[k*2+1], D[k*2+2])


// [a, b]
findMin(a, b)
    return query(a, b+1, 0, 0, n)


// [a, b)
query(a, b, k, l, r)
    if r <= a || b <= l:   // case 1
        return INF
    if a <= l && r <= b:   // case 2
        return D[k]

    // case 3
    vl = query(a, b, k*2 + 1, l, (l + r)/2)
    vr = query(a, b, k*2 + 2, (l + r)/2, r)
    return min(vl, vr)
```

5 Lazy Evaluation

To implement an efficient segment tree, for some cases (e.g. RMQ + RUQ), we need a technique called **lazy evaluation**. In addition to `D[i]`, let `lazy[i]` be a value which is equally updated (or added in case of range add) for the corresponding segment but has not been reflected to `D[i]`. At the appropriate time, `lazy[k]` is expanded by the following procedure (for RMQ).

```
lazy_evaluate(k)
    if lazy[k] == NUL: return
    D[k] = lazy[k]
    if k < n - 1:  // not leaf
        lazy[2*k + 1] = lazy[k]
        lazy[2*k + 2] = lazy[k]
    lazy[k] = NUL
```

`lazy_evaluate(k)` should be activated before the corresponding segment which is represented by `k`, is updated or queried.

You can implement the query function with the lazy evaluation as follows (for RMQ).

```
query(a, b, k, l, r)
    if r <= a || b <= l:   // case 1
        return INF

    lazy_evaluate(k)

    if a <= l && r <= b:   // case 2
        return D[k]

    // case 3
    vl = query(a, b, k*2 + 1, l, (l + r)/2)
    vr = query(a, b, k*2 + 2, (l + r)/2, r)
    return min(vl, vr)
```

You can implement the update function with the lazy evaluation as follows (for RMQ).

```
update(a, b, k, l, r, x)
    lazy_evaluate(k)
    if r <= a || b <= l:
        return
    if a <= l && r <= b:
        lazy[k] = x
    else
        update(a, b, k*2+1, l, (l+r)/2, x)
        update(a, b, k*2+2, (l+r)/2, r, x)
        D[k] = min(D[k*2+1], D[k*2+2])

    lazy_evaluate(k)
```

In this implementation, you should perform the lazy evaluation at both the beginning and the end of the update function. The first one is to propagate the lazy evaluation. On the other hand, we need the second one to use reflected values of `D[k*2+1]` and `D[k*2+2]`.

6 Assignments

Place

https://onlinejudge.u-aizu.ac.jp/beta/room.html#CSC11F_2023_Week_03

Duration

2 week

Problem A: Range Minimum Query (RMQ)

Write a program which manipulates a sequence $A = \{a_0, a_1, ..., a_{n-1}\}$ with the following operations:

- find(s, t): report the minimum element in $a_s, a_{s+1}, ..., a_t$.
- update(i, x): change a_i to x.

Note that the initial values of a_i $(i = 0, 1, ..., n-1)$ are $2^{31} - 1$.

Input

n q
com_0 x_0 y_0
com_1 x_1 y_1
...
com_{q-1} x_{q-1} y_{q-1}

In the first line, n (the number of elements in A) and q (the number of queries) are given. Then, q queries are given where com represents the type of queries. '`0`' denotes update(x_i, y_i) and '`1`' denotes find(x_i, y_i).

Output

For each find operation, print the minimum element.

Constraints

- $1 \leq n, q \leq 100,000$
- If com_i is 0, then $0 \leq x_i < n$, $0 \leq y_i < 2^{31} - 1$
- If com_i is 1, then $0 \leq x_i < n$, $0 \leq y_i < n$.

Sample Input 1

```
3 5
0 0 1
0 1 2
0 2 3
1 0 2
1 1 2
```

Sample Output 1

```
1
2
```

Sample Input 2

```
1 3
1 0 0
0 0 5
1 0 0
```

Sample Output 2

```
2147483647
5
```

Problem B: Range Update Query (RUQ)

Write a program which manipulates a sequence $A = \{a_0, a_1, ..., a_{n-1}\}$ with the following operations:

- update(s, t, x): change $a_s, a_{s+1}, ..., a_t$ to x.
- find(i): output the value of a_i.

Note that the initial values of a_i $(i = 0, 1, ..., n-1)$ are $2^{31} - 1$.

Input

n q
$query_1$
:
$query_q$

In the first line, n (the number of elements in A) and q (the number of queries) are given. Then, ith query $query_i$ is given in the following format:

0 s t x

or

1 i

The first digit represents the type of the query. '0' denotes update(s, t, x) and '1' denotes find(i).

Output

For each $find$ operation, print the value.

Constraints

- $1 \leq n, q \leq 100,000$
- $0 \leq s \leq t < n$
- $0 \leq i < n$
- $0 \leq x < 2^{31} - 1$

Sample Input 1

```
3 5
0 0 1 1
0 1 2 3
0 2 2 2
1 0
1 1
```

Sample Output 1

```
1
3
```

Sample Input 2

```
1 3
1 0
0 0 0 5
1 0
```

Sample Output 2

```
2147483647
5
```

Problem C: RMQ and RUQ

Write a program which manipulates a sequence $A = \{a_0, a_1, ..., a_{n-1}\}$ with the following operations:

- update(s, t, x): change $a_s, a_{s+1}, ..., a_t$ to x.
- find(s, t): report the minimum element in $a_s, a_{s+1}, ..., a_t$.

Note that the initial values of a_i $(i = 0, 1, ..., n-1)$ are $2^{31} - 1$.

Input

n q
$query_1$
:
$query_q$

In the first line, n (the number of elements in A) and q (the number of queries) are given. Then, ith query $query_i$ is given in the following format:

0 s t x

or

1 s t

The first digit represents the type of the query. '0' denotes update(s, t, x) and '1' denotes find(s, t).

Output

For each $find$ operation, print the minimum value.

Constraints

- $1 \leq n, q \leq 100,000$
- $0 \leq s \leq t < n$
- $0 \leq x < 2^{31} - 1$

Sample Input 1

```
3 5
0 0 1 1
0 1 2 3
0 2 2 2
1 0 2
1 1 2
```

Sample Output 1

```
1
2
```

Sample Input 2

```
1 3
1 0 0
0 0 0 5
1 0 0
```

Sample Output 2

```
2147483647
5
```