

Segment Tree | Set 2 (Range Minimum Query)

We have introduced [segment tree with a simple example](#) in the previous post. In this post, [Range Minimum Query](#) problem is discussed as another example where Segment Tree can be used. Following is problem statement.

We have an array $arr[0 \dots n-1]$. We should be able to efficiently find the minimum value from index qs (query start) to qe (query end) where $0 \leq qs \leq qe \leq n-1$. The array is static (elements are not deleted and inserted during the series of queries).

A **simple solution** is to run a loop from qs to qe and find minimum element in given range. This solution takes $O(n)$ time in worst case.

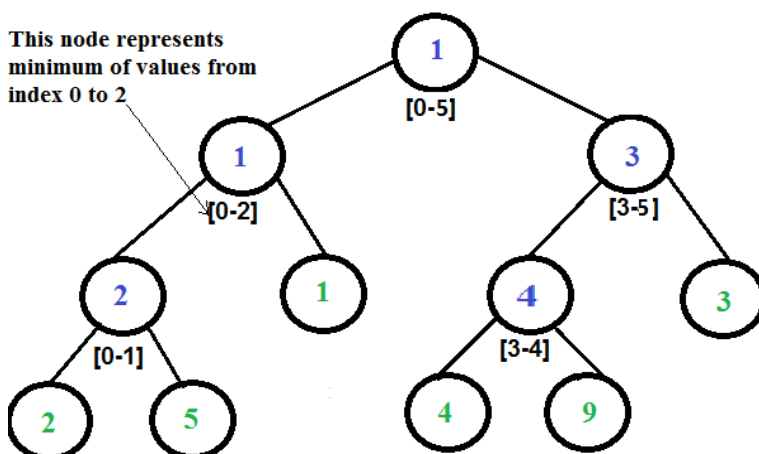
Another solution is to create a 2D array where an entry $[i, j]$ stores the minimum value in range $arr[i..j]$. Minimum of a given range can now be calculated in $O(1)$ time, but preprocessing takes $O(n^2)$ time. Also, this approach needs $O(n^2)$ extra space which may become huge for large input arrays.

Segment tree can be used to do preprocessing and query in moderate time. With segment tree, preprocessing time is $O(n)$ and time to for range minimum query is $O(\log n)$. The extra space required is $O(n)$ to store the segment tree.

Representation of Segment trees

1. Leaf Nodes are the elements of the input array.
2. Each internal node represents minimum of all leaves under it.

An array representation of tree is used to represent Segment Trees. For each node at index i , the left child is at index $2*i+1$, right child at $2*i+2$ and the parent is at $\lfloor (i-1)/2 \rfloor$.



Segment Tree for input array {2, 5, 1, 4, 9, 3}

Construction of Segment Tree from given array

We start with a segment $arr[0 \dots n-1]$. and every time we divide the current segment into two halves(if it has not yet become a segment of length 1), and then call the same procedure on both halves, and for each such segment, we store the minimum value in a segment tree node. All levels of the constructed segment tree will be completely filled except the last level. Also, the tree will be a **Full Binary Tree** because we always divide segments in two halves at every level. Since the constructed tree is always full binary tree with n leaves, there will be $n-1$ internal nodes. So total number of nodes will be $2*n - 1$.

Height of the segment tree will be $\lceil \log_2 n \rceil$. Since the tree is represented using array and relation between parent and child indexes must be maintained, size of memory allocated for segment tree will be $2 * 2^{\lceil \log_2 n \rceil} - 1$.

Query for minimum value of given range

Once the tree is constructed, how to do range minimum query using the constructed segment tree. Following is algorithm to get the minimum.

```
// qs --> query start index, qe --> query end index
int RMQ(node, qs, qe)
{
    if range of node is within qs and qe
        return value in node
    else if range of node is completely outside qs and qe
        return INFINITE
    else
        return min( RMQ(node's left child, qs, qe), RMQ(node's right child, qs, qe) )
}
```

Implementation:

```
// Program for range minimum query using segment tree
#include <stdio.h>
#include <math.h>
#include <limits.h>

// A utility function to get minimum of two numbers
int minVal(int x, int y) { return (x < y)? x: y; }

// A utility function to get the middle index from corner
int getMid(int s, int e) { return s + (e -s)/2; }

/* A recursive function to get the minimum value in a given range
   indexes. The following are parameters for this function:

   st    --> Pointer to segment tree
   index --> Index of current node in the segment tree.
             passed as root is always at index 0
   ss & se --> Starting and ending indexes of the segment currently
               stored in st[index]
   qs & qe --> Starting and ending indexes of query range
int RMQutil(int *st, int ss, int se, int qs, int qe, int index)
```

```

{
    // If segment of this node is a part of given range,
    // min of the segment
    if (qs <= ss && qe >= se)
        return st[index];

    // If segment of this node is outside the given range
    if (se < qs || ss > qe)
        return INT_MAX;

    // If a part of this segment overlaps with the given range
    int mid = getMid(ss, se);
    return minVal(RMQUtil(st, ss, mid, qs, qe, 2*index+1,
                          RMQUtil(st, mid+1, se, qs, qe, 2*index+2));
}

// Return minimum of elements in range from index qs (query start)
// qe (query end). It mainly uses RMQUtil()
int RMQ(int *st, int n, int qs, int qe)
{
    // Check for erroneous input values
    if (qs < 0 || qe > n-1 || qs > qe)
    {
        printf("Invalid Input");
        return -1;
    }

    return RMQUtil(st, 0, n-1, qs, qe, 0);
}

// A recursive function that constructs Segment Tree for
// arr[ss..se]. si is index of current node in segment tree st
int constructSTUtil(int arr[], int ss, int se, int *st, int si)
{
    // If there is one element in array, store it in current
    // segment tree and return
    if (ss == se)
    {
        st[si] = arr[ss];
        return arr[ss];
    }

    // If there are more than one elements, then recur for
    // left and right subtrees and store the minimum of two values
    int mid = getMid(ss, se);
    st[si] = minVal(constructSTUtil(arr, ss, mid, st, si),
                    constructSTUtil(arr, mid+1, se, st, si));
    return st[si];
}

/* Function to construct segment tree from given array. It
   allocates memory for segment tree and calls constructSTUtil()
   to fill the allocated memory */
int *constructST(int arr[], int n)
{
    // Allocate memory for segment tree

```

```

int x = (int)(ceil(log2(n))); //Height of segment tree
int max_size = 2*(int)pow(2, x) - 1; //Maximum size of segment tree
int *st = new int[max_size];

// Fill the allocated memory st
constructSTUtil(arr, 0, n-1, st, 0);

// Return the constructed segment tree
return st;
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 3, 2, 7, 9, 11};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Build segment tree from given array
    int *st = constructST(arr, n);

    int qs = 1; // Starting index of query range
    int qe = 5; // Ending index of query range

    // Print minimum value in arr[qs..qe]
    printf("Minimum of values in range [%d, %d] is = %d\n",
           qs, qe, RMQ(st, n, qs, qe));

    return 0;
}

```

Output:

```
Minimum of values in range [1, 5] is = 2
```

Time Complexity:

Time Complexity for tree construction is $O(n)$. There are total $2n-1$ nodes, and value of every node is calculated only once in tree construction.

Time complexity to query is $O(\log n)$. To query a range minimum, we process at most two nodes at every level and number of levels is $O(\log n)$.

Please refer following links for more solutions to range minimum query problem.

<http://community.topcoder.com/tc?>

[module=Static&d1=tutorials&d2=lowestCommonAncestor#Range_Minimum_Query_\(RMQ\)](http://www.geogebra.org/m/Static&d1=tutorials&d2=lowestCommonAncestor#Range_Minimum_Query_(RMQ))

http://wcipeg.com/wiki/Range_minimum_query