Practical 6

# Student Details

* Name : Tushar Harsora
* Roll : 19BCE509

# Aim

Implement the segment trees using arrays and give the query function to operate in log(n) time. The combine function here is “min” function.

# Code

* SegmentTree.h (Header File)

#pragma once

#include <vector>

#include <cstdint>

template<class T, typename Combine>

struct SegmentTree {

private:

using index = int64\_t;

std::vector<T> m\_arr{};

Combine m\_comb{};

size\_t m\_size;

const T dummy;

public:

SegmentTree(const std::vector<T>& vec, const T& dummy) : dummy(dummy) {

auto n = vec.size();

m\_size = vec.size();

m\_arr.resize(m\_size \* 2);

for (size\_t i = 0; i < n; i++) {

m\_arr[n + i] = vec[i];

}

for (size\_t i = n - 1; i > 0; i--) { // One Based Indexing

m\_arr[i] = m\_comb(m\_arr[i \* 2], m\_arr[i \* 2 + 1]);

}

}

T query(index l, index r) {

T ans = dummy;

for (l += m\_size, r += m\_size; l <= r; l = (l + 1) / 2, r = (r - 1) / 2) {

if (l & 1) {

ans = m\_comb(ans, m\_arr[l]);

}

if (!(r & 1)) {

ans = m\_comb(ans, m\_arr[r]);

}

}

return ans;

}

};

* Segment Tree Test (main function)

#include "SegmentTree.h"

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct min\_fun {

int operator()(const int& lhs, const int& rhs) {

return std::min(lhs, rhs);

}

};

int main() {

int n;

cin >> n;

vector<int> arr(n);

for (auto& ele : arr)

cin >> ele;

SegmentTree<int, min\_fun> s(arr, INT\_MAX);

int q;

cin >> q;

while (q--) {

int x, y;

cin >> x >> y;

cout << s.query(x, y) << endl;

}

return 0;

}

# Inputs

5

5 3 2 1 4

5

0 1

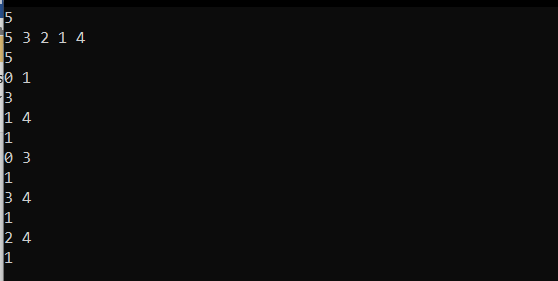
1 4

0 3

3 4

2 4

# Screenshots of output



# Conclusion

Here We learned about how to make space efficient implementation of the segment tree and how to use combine function. And do time efficient range queries.