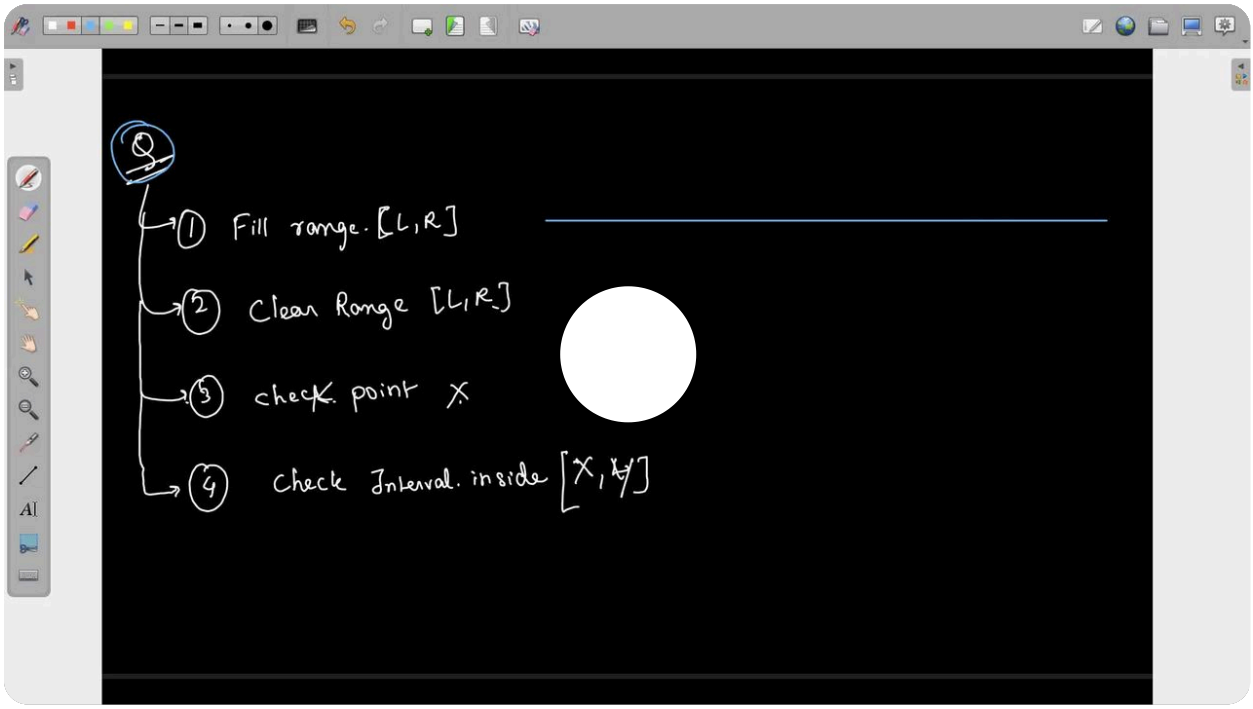
Range Maintenance Ideas - 2



Range Maintenance Ideas - 2

Problem Statement:

We are given blank number line. And then We have to perform Q queries over it. Each query can be of 4 different types:

1. Fill range [L,R] - A line will cover all the points between [L, R]
2. Clear Range [L,R] - remove all ranges passing through [L, R]
3. Check Point X - if it is active or not, by active means range pass through that or not
4. Check Interval inside [X,Y] - if any point inside range [X,Y] are active inside or not

Approach:

We divide the problem into 5 different test cases which cover whole problem and try to solve each one by one.

To solve the problem of range maintenance on a number line efficiently, we can use a set data structure to store the active ranges. Each range `[L, R]` can be represented as a pair `(L, R)` and stored in the set.

1. **Data Structure:** Use a set to store active ranges. Each range will be represented by a pair `(L, R)` where `L` is the left endpoint and `R` is the right endpoint.

2. **Filling Ranges:**
    - When filling a range `[L, R]`, check if it overlaps with any existing ranges in the set.
    - If it overlaps, merge it with the overlapping ranges to form a single contiguous range.
    - Remove any ranges fully covered by the new range.
    - Insert the new merged range into the set.
3. **Clearing Ranges:**
    - When clearing a range `[L, R]`, adjust existing ranges as necessary:
        - Split any ranges that intersect with the clearing range into smaller ranges.
        - Remove any ranges fully contained within the clearing range.
    - Remove the clearing range itself from the set.
4. **Checking Point Activity:**
    - To check if a point `X` is active, search for the range containing `X` in the set:
        - If found, the point is active; otherwise, it's inactive.
5. **Checking Interval Activity:**
    - To check if any or all points within an interval `[X, Y]` are active:
        - Search for any active ranges intersecting the interval `[X, Y]`.
        - If any active ranges are found, return true for `check_range_any`.
        - For `check_range_all`, additionally check if the interval is fully covered by active ranges.

Algorithm:

1. **Fill Range Algorithm (`fill_range(L, R)`):**

1. Find the first range in the set with a right endpoint greater than or equal to `L`.
2. Merge the new range `[L, R]` with any overlapping ranges found.
3. Remove any ranges fully covered by the merged range.
4. Insert the merged range into the set.

2. **Clear Range Algorithm (`clear_range(L, R)`):**

    1. Find and remove any ranges fully contained within `[L, R]`.
    2. Adjust any overlapping ranges:
        - Split them into smaller ranges if they intersect with `[L, R]`.
        - Remove any parts of the overlapping ranges fully contained within `[L, R]`.

3. **Check Point Activity Algorithm (`check_point(X)`):**

    1. Search for the range containing `X` in the set.
    2. If found, return true; otherwise, return false.

4. **Check Interval Activity Algorithms (`check_range_any(X, Y)` and `check_range_all(X, Y)`):**

    1. Search for any active ranges intersecting the interval `[X, Y]` in the set.
    2. If any ranges are found:
        - For `check_range_any`, return true.
        - For `check_range_all`, check if the interval `[X, Y]` is fully covered by active ranges. If so, return true; otherwise, return false.

Code:

Copy

```
using ii = pair<int, int>;
#define F first
#define S second

struct range_maintainance
{
    set<ii> st;
    void fill_range(int l, int r)
    {
        auto it = st.upper_bound({l, 1e9});
        if (it != st.begin())
        {
            it--;
            if (it->second >= l)
            {
                l = it->first;
                r = max(r, it->second);
                st.erase(it);
            }
        }

        auto it = st.upper_bound({r, 1e9});
        if (it != st.begin())
        {
            it--;
            if (it->second >= r)
            {
                r = it->second;
                l = min(r, it->second);
                st.erase(it);
            }
        }
```

```
    while (1)
    {
        it = st.lower_bound({l, 0});
        if (it == st.end() || (it->second <= r))
        {
            break;
        }
        else
        {
            st.erase(it);
        }
    }

    st.insert({l, r});
}

void clear_range(int l, int r)
{
    auto it = st.upper_bound({l, 1e9});
    if (it != st.begin())
    {
        it--;
        if (it->second >= r)
        {
            int lo1 = it->first;
            int hi1 = l;

            int lo2 = r;
            int hi2 = it->second;

            st.erase(it);
            st.insert({lo1, hi1});
            st.insert({lo2, hi2});
            return;

            if (it->second >= l)
            {
                int lo = it->first;
                int hi = l;
                st.erase(it);
                st.insert({lo, hi});
            }
        }
    }

    it = st.upper_bound({r, 1e9});
    if (it != st.begin())
    {
        it--;
        if (it->second >= r)
        {
            int lo = r;
            int hi = it->second;
            st.erase(it);
            st.insert({lo, hi});
        }
    }

    while (1)
    {
        it = st.lower_bound({l, 0});
        if (it = st.end() || (it->first > r))
        {
            break;
```

```
        }
        else
        {
            st.erase(it);
        }
    }
}

bool check_point(int x)
{
    auto it = st.upper_bound({x, 1e9});
    if (it == st.begin())
    {
        return 0;
    }
    else
    {
        it--;
        if (it->second >= x)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}

bool check_range_any(int x, int y)
{
    auto it = st.upper_bound({x, 1e9});
    if (it != st.end())
    {
        if (it->first <= y)
        {
            return 1;
        }
    }
    return check_point(x);
}

bool check_range_all(int x, int y)
{
    auto it = st.upper_bound({x, 1e9});
    if (it == st.begin())
    {
        return 0;
    }
    else
    {
        it--;
        if (it->second >= y)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}
```

```
};
```

Time Complexity

- In general, the time complexity of most operations is O(log N) or O(N), where N is the number of active ranges in the set.
- The overall performance can be affected by the distribution of ranges and the frequency of overlapping or fully contained ranges.

Write a comment