**IMP NOTES**

**1. Incorrect usage of str.find():**

* The find() function returns the **index** of the first occurrence of the character if found, or std::string::npos (not '\0') if the character is not found.
* You should check for std::string::npos, not '\0'.

**2. String range in s.substr(i, j):**

* The s.substr(i, j) takes two arguments: the starting position i and the length of the substring, not the ending position j. To get a substring between i and j-1, you should use s.substr(i, j-i).

**3. Maximization logic inside the loop:**

* The maximization logic (maxi = max(maxi, j - i + 1);) should be placed outside the inner while loop.

**4. Substring creation on each iteration:**

* Continuously creating substrings inside the loop is inefficient. Instead, a more efficient approach is to use a set to track unique characters in the window and slide the window using two pointers.

## 1. **Operations on** std::set

A set is an ordered container that contains unique elements. Here's a list of operations you can perform on std::set.

### Basic Operations

cpp

Copy code

std::set<int> mySet;

* **Insert Elements**:
  + mySet.insert(value); – Inserts a value into the set if it doesn’t already exist.
* **Erase Elements**:
  + mySet.erase(value); – Erases the value from the set if it exists.
  + mySet.erase(iterator); – Erases the element at the iterator position.
  + mySet.clear(); – Removes all elements from the set.
* **Find Elements**:
  + mySet.find(value); – Returns an iterator to the element, or mySet.end() if not found.
  + mySet.count(value); – Returns 1 if the value exists in the set, otherwise 0.
* **Check if Empty**:
  + mySet.empty(); – Returns true if the set is empty, otherwise false.
* **Size of Set**:
  + mySet.size(); – Returns the number of elements in the set.
* **Range Operations**:
  + mySet.lower\_bound(value); – Returns an iterator to the first element that is not less than the given value.
  + mySet.upper\_bound(value); – Returns an iterator to the first element that is greater than the given value.
  + mySet.equal\_range(value); – Returns a pair of iterators representing the range of elements equal to the given value.

### Iterating through the Set

cpp

Copy code

for (auto it = mySet.begin(); it != mySet.end(); ++it) {

std::cout << \*it << " ";

}

### Other Operations

* **Swapping**:
  + mySet.swap(anotherSet); – Swaps the content of mySet with another set.
* **Comparison**:
  + mySet == anotherSet; – Compares two sets for equality.
  + mySet != anotherSet; – Compares two sets for inequality.
  + mySet < anotherSet;, mySet > anotherSet; – Lexicographical comparison of two sets.

## 2. **Operations on** std::map

A map is an ordered container that stores key-value pairs, with unique keys.

cpp

Copy code

std::map<int, std::string> myMap;

### Basic Operations

* **Insert Elements**:
  + myMap.insert({key, value}); – Inserts a key-value pair.
  + myMap[key] = value; – Inserts or updates the value for the key.
* **Access Elements**:
  + myMap[key]; – Returns the value associated with the key. Inserts key with default value if it doesn’t exist.
  + myMap.at(key); – Returns the value for the key if it exists, otherwise throws an exception.
* **Erase Elements**:
  + myMap.erase(key); – Erases the element with the given key.
  + myMap.erase(iterator); – Erases the element at the iterator position.
  + myMap.clear(); – Removes all elements from the map.
* **Find Elements**:
  + myMap.find(key); – Returns an iterator to the element with the given key, or myMap.end() if not found.
  + myMap.count(key); – Returns 1 if the key exists, otherwise 0.
* **Check if Empty**:
  + myMap.empty(); – Returns true if the map is empty, otherwise false.
* **Size of Map**:
  + myMap.size(); – Returns the number of elements in the map.
* **Range Operations**:
  + myMap.lower\_bound(key); – Returns an iterator to the first element that is not less than the given key.
  + myMap.upper\_bound(key); – Returns an iterator to the first element that is greater than the given key.
  + myMap.equal\_range(key); – Returns a pair of iterators representing the range of elements equal to the given key.

### Iterating through the Map

cpp

Copy code

for (auto it = myMap.begin(); it != myMap.end(); ++it) {

std::cout << it->first << ": " << it->second << std::endl;

}

### Other Operations

* **Swapping**:
  + myMap.swap(anotherMap); – Swaps the content of myMap with another map.
* **Comparison**:
  + myMap == anotherMap; – Compares two maps for equality.
  + myMap != anotherMap; – Compares two maps for inequality.
  + myMap < anotherMap;, myMap > anotherMap; – Lexicographical comparison of two maps.

## Summary of Key Operations

### Common Operations (set and map)

* **Insert**: insert(value) or insert({key, value})
* **Erase**: erase(value) or erase(key)
* **Find**: find(value) or find(key)
* **Check empty**: empty()
* **Size**: size()
* **Lower Bound**: lower\_bound(value) or lower\_bound(key)
* **Upper Bound**: upper\_bound(value) or upper\_bound(key)
* **Range**: equal\_range(value) or equal\_range(key)
* **Clear**: clear()
* **Swap**: swap(other)

### Unique to set

* count(value) to check if a value is present.

### Unique to map

* at(key) to access a value safely.
* operator[] for insertion and access by key.