C++ - Standard Template Library(STL)

SoCP - Lecture 2
ALGORITHMS AND CODING CLUB IIT DELHI
Himanshu Gaurav Singh

std::set

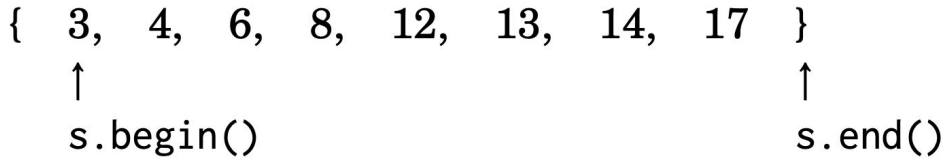
- The set in C++ - STL is basically the heap data structure.
- It is a set of values(strictly speaking, objects that can be uniquely ordered somehow) in which you can insert and delete in O(log n) worst case time complexity, along with O(log n) time to search for a value in it.
- This is a major improvement over storing a collection of values in some other basic way. For example an array, which will require O(n) time to search.
- Note that no two elements in a set are same.

Attribute functions for set

Suppose we have a set S.

1. S.insert(x) inserts the value x into the set S if it is already not present, else it does nothing. Time Complexity - O(log n)
2. S.erase(x) erases the value x from the set S if it is present, else it does nothing. Time Complexity - O(log n). Here x can even be an iterator too.

Iterators in std::set



1. S.begin() - returns the iterator pointing at the first (smallest) element in the set.
2. S.end() - returns an iterator pointing at a location just after the last(largest) element in the set.(Note that this way of explanation is just to make the cp implementation intuitive)
3. S.rbegin() - returns an iterator pointing to the element at the "beginning from right", or the last element of the last.
4. S.rend() - returns an iterator pointing to the element at the the "last from right", or one beyond the first element of the last.
5. The "++" operator on the <iterator :: set<?>> returns the successor of the operand iterator. Similarly "--" operator gives the predecessor.
6. S.find(x) returns the iterator pointing to x if x is present in the set, else it returns S.end()

Caution while working with iterators

- While iterating through an STL container using iterators, do not insert or delete elements from it. This will cause undefined behaviour if done.
- This happens largely because STL containers are dynamically allocated memory objects and the memory locations they reside changes if the size of the container increases/decreases.
- You can only use ++ or -- operator, you cannot assign arbitrary values to an iterator or do arithmetic on it like it= it + 1 ;

Std::set for non primitive objects

1. How to proceed if you need a set of objects that are not simple integers or stringers but a combination of two or more of them ?

   Ans: Modify the definition of “<” using operator overloading for that particular datatype.

std::multiset

1. This is similar to std::set but can have non-distinct values in it.
2. Keep in mind that S.erase(x) erases all instances of x in the multiset. To erase a single instance, you might do S.erase(S.find(x))
3. Other properties are similar to std::set

std::unordered_set

- This is basically an implementation of hashing. This provides O(1) search and delete avg case time complexity but can go upto O(n) (if the hash functions are not very well formed), though this happens rarely.
- The iterators and stuff are similar to that in std::set.

std::map

- While set is a collection of values, map is basically a collection of (key,value) pairs. Similar to a python dictionary.
- For a map M, you can access a value corresponding to a key using M[key]. Note that the key and value can be any data type(not necessarily same).

Iterators in std::map

- The iterators in a map work similar to that in set.
- You can go through the elements of a map by a “for-loop” in the following way

std::unordered_map

1. This again is basically the implementation of a hash map. It has search and delete time complexities of O(1) average and O(n) worst case.
2. The iterators and stuff are similar to that off as map