C++ STL

Srivaths P

Goal

To review:

- Pair, Vector, Set, Map

To learn:

- Stack, Queue, Deque, Priority Queue
- Common STL functions, binary search on datatypes.

Pair

Pairs are very useful when dealing with two related values. For example, storing a range [L, R].

Pairs have inbuilt comparators such as <, >, etc.

Usage:

```
pair<int, int> p = {1, 7};
cout << p.first << endl; // outputs 1
cout << p.second << endl; // outputs 7
```

Vector

Vectors store an ordered collection of data. Unlike arrays, vectors can be resized. They also have far more features than arrays.

Useful vector functions:

```
v.begin(), v.end();
v.rbegin(), v.rend();
v.push_back(val), v.pop_back();
v.empty(), v.size();
v.insert(it, val), v.erase(it);
v.clear();
```

Set

Sets are used to store values without indexing.

- Sets store unique values in a sorted order.
  Search, removal, insertion of an element is O(log N).

- Unordered Sets store unique values, in any order.
  Search, removal, insertion of an element is O(1).

- Multisets can store multiple values in a sorted order.
  Search, removal, insertion of an element is O(log N).

- Unordered multisets store multiple values, in any order.
  Search, removal, insertion of an element is O(1).

Map

Maps are similar to an array, where index can be anything.

- Maps store unique keys in sorted order.
  Search, removal, insertion of an element is O(log N).

- Unordered Maps store unique keys, in any order.
  Search, removal, insertion of an element is O(1).

- Multimaps can store multiple keys in a sorted order.
  Search, removal, insertion of an element is O(log N).

- Unordered multimaps store multiple keys, in any order.
  Search, removal, insertion of an element is O(1).

Custom comparators for set/map

For set/map to work for some datatype, the datatype must have "<" function implemented.

Otherwise, it must have a custom comparator passed to the declaration as follows:

```
set<int, decltype(cmp)*> s(cmp);
```

Where `cmp` is the custom comparator.

The same syntax follows for map as well.

Stack

Stack is a *container adapter* that uses LIFO.

They can only push at the end and pop from the end. Stacks do not support indexing.

Only useful stack operations:

```
s.size(), s.empty()
s.push(), s.pop()
s.top()
```

Queue

Queue are very similar to stacks, except they use FIFO instead of LIFO.

Only useful queue operations:

```
q.size(), q.empty()
q.push(), q.pop()
q.front()
```

Both stack and queue use deque as default container

Deque

Deque is very similar to vectors, but it supports insertion and deletion of elements from both sides of the deque.

Deque functions (excluding vector functions):

```
d.push_front();
d.pop_front();
```

Deques are marginally slower than vectors in terms of performance.

Priority Queue

Priority queue is similar to queue, except that the popped item will be sorted in increasing order.

It takes O(log N) time to push and pop elements.

Priority queue can store duplicates, similar to multiset.

Priority Queue

Indexing is impossible in priority_queue, and binary search cannot be performed on it.

Priority queues are faster than sets as they have a lower constant factor.

Syntax:

```
priority_queue<T, vector<T>, decltype(&cmp)> pq(cmp);
```

STL functions

STL functions on containers usually perform some algorithm on an iterator, or a range [L, R) where L and R are iterators.

The functions might also need a custom function. For example, custom comparators are passed as functions.

They might also need input of a value. For example, if we are looking for an element, we need to enter the target as a parameter.

Useful STL functions

- sort
- min_element, max_element
- reverse
- find, count
- fill, iota
- unique, accumulate
- is_sorted

STL binary search function

The STL binary search functions are:

- binary_search: Returns a bool denoting whether an element is present or not
- lower_bound: Returns the iterator of the first element greater or equal to the given target
- upper_bound: Returns the iterator of the first element greater than the given target

The syntax for all of them is similar to:

```
function(begin_it, end_it, target, cmp);
```

Binary search on sorted datatypes

When a datatype is sorted by default the binary search functions are in-built into the datatype.

```
auto it = sorted_type.lower_bound(target);
```

Always prefer the in-built version opposed to the STL functions when *random access* is not possible, as the time complexity is likely to be better.

Note that the comparator is taken as the provided comparator. It cannot be modified.

Problem Solving

Challenge problems:

1. https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/
2. https://leetcode.com/problems/maximum-nesting-depth-of-the-parentheses/

Problem Solving:

https://leetcode.com/problems/valid-parentheses/
https://leetcode.com/problems/min-stack/
https://codeforces.com/problemset/problem/1345/B

Challenge Problems:

https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/
https://leetcode.com/problems/maximum-nesting-depth-of-the-parentheses/
https://leetcode.com/problems/kth-largest-element-in-a-stream/
https://codeforces.com/contest/1277/problem/B

Resources:

https://www.cppreference.com/Cpp_STL_ReferenceManual.pdf

https://devdocs.io/cpp/container (for STL containers)

https://devdocs.io/cpp/algorithm (for STL algorithms)

https://stackoverflow.com/questions/2620862/using-custom-stdset-comparator (For custom set comparators)

Try to learn about various other algorithms such as transform, rotate, etc.

Resources:

- https://baptiste-wicht.com/posts/2012/12/cpp-benchmark-vector-list-deque.html

  Comparision of time taken for different datatypes

- https://devdocs.io/cpp/algorithm/lower_bound
  https://devdocs.io/cpp/algorithm/upper_bound

  Binary search functions

- https://codeforces.com/blog/entry/11080
  PBDS (Policy Based Data Structure)

- https://codeforces.com/blog/entry/62393 (For hashing)

Thanks for watching!