Session 1: Core Containers and Basics

Mohamed Emad Mahrous

2nd of November, 2024.

Introduction to STL

The Standard Template Library (STL) is a powerful and essential component of the C++ Standard Library that provides a set of common data structures and algorithms. It includes several key features that make it particularly significant in both competitive programming and software development:

1. **Key Components of STL**
   - **Containers:** These are data structures that store collections of objects. Common containers include:
     - Vector: A dynamic array that can resize itself.
     - List: A doubly linked list.
     - Deque: A double-ended queue.
     - Set: A collection of unique elements.
     - Map: A collection of key-value pairs, where keys are unique.
   - **Algorithms:** STL provides a wide array of algorithms that operate on containers, such as:
     - Sorting (e.g., sort, stable_sort)
     - Searching (e.g., binary_search, find)
     - Modifying (e.g., reverse, shuffle)
     - Numeric (e.g., accumulate, inner_product)
   - **Iterators:** These are objects that allow traversal of the elements in a container, similar to pointers. STL provides different types of iterators, such as:
     - Input Iterators: For reading data.
     - Output Iterators: For writing data.
     - Forward Iterators: For traversing in one direction.
     - Bidirectional Iterators: For traversing in both directions.
     - Random Access Iterators: For accessing elements directly (like pointers).
   - **Significance in Competitive Programming**
     - **Efficiency:** STL's algorithms are often optimized for performance. Using these algorithms can save time in implementation and can lead to more efficient solutions.
     - **Time-Saving:** STL provides ready-to-use data structures and algorithms, allowing competitors to focus on problem-solving rather than on writing and debugging complex data structures from scratch.
     - **Versatility:** The range of data structures and algorithms available in STL makes it easier to handle various types of problems, from simple to complex.
     - **Debugging:** STL components are well-tested and can reduce the chances of bugs, which is crucial during time-constrained competitions.

2. **Sequential Containers**

   - A vector in C++ is a dynamic array that can grow and shrink in size. It is part of the Standard Template Library (STL) and provides many useful functions.

   Key Functions

   Here are some of the most commonly used functions with vectors:

   - **push_back(value)**: Adds an element to the end of the vector.
   - **pop_back()**: Removes the last element from the vector.
   - **size()**: Returns the number of elements in the vector.
   - **back()**: Returns the last element in the array

   Common Scenarios

   Vectors can be used in various scenarios, including:

   - Storing a list of items, such as student grades or names.
   - Implementing data structures like stacks and queues.
   - Dynamic arrays where the size needs to change based on user input.

3. **Associative Containers**

   - A set in C++ is a collection of unique elements, which is part of the Standard Template Library (STL). It automatically manages duplicates and is typically implemented as a balanced binary search tree.

   Key Functions

   Here are some essential functions associated with sets:

   - **insert(value)**: Adds a new element to the set.
   - **find(value)**: Searches for an element in the set and returns an iterator to it, or the end iterator if not found.
   - **count(value)**: Returns the number of occurrences of an element in the set (should be 0 or 1 for a set).
   - **erase(value)**: Removes an element from the set.

   Common Scenarios

   - **Unique Item Storage:** Sets are ideal for storing collections of unique items, such as usernames, email addresses, or product IDs. This ensures that no duplicates are allowed, which is useful for validating input data or maintaining distinct lists.
   - **Membership Testing:** Sets provide efficient membership testing. When you need to frequently check whether an item exists in a collection (like checking if a user is on a blacklist), sets can perform this operation in average O(1)O(1) time complexity.
   - **Set Operations:** Sets are useful for performing mathematical set operations, such as unions, intersections, and differences. For example, if you have two sets of data and need to find common elements (intersection) or combine them without duplicates (union), sets provide built-in functionalities for these operations.

- **Removing Duplicates:** When processing a list of items, sets can automatically remove duplicates. For example, if you have a list of survey responses and want to identify unique answers, inserting them into a set will filter out duplicates.
- **Collecting Results:** In algorithms that need to gather results from different sources, like finding unique elements across multiple datasets, using a set can simplify the process of aggregating and ensuring that only unique results are kept.
- **Dynamic Programming:** Sets can be helpful in dynamic programming scenarios where the solution to a problem depends on unique combinations of items or states, such as finding unique sums or combinations of values.

The second section

- A map in C++ is a collection of key-value pairs, where each key is unique. It is part of the Standard Template Library (STL) and allows for efficient retrieval and modification of values based on their keys.

Key Operations

Here are some essential operations you can perform on maps:

- **insert(key, value)**: Adds a new key-value pair to the map.
- **find(key)**: Searches for a key in the map and returns an iterator to it, or the end iterator if not found.
- **erase(key)**: Removes the key-value pair from the map.

Common Scenarios

- **Counting Frequencies:** Maps are frequently used to count occurrences of elements in an array or list. For example, you can count how many times each character appears in a string or how many times each number appears in an array. This is often useful in problems requiring frequency analysis.
- **Tracking Distinct Elements:** In problems where you need to maintain a count of distinct elements while processing a sequence, maps can help keep track of the number of unique values seen so far, facilitating operations that require uniqueness.
- **Mapping Relationships:** For problems that involve relationships or mappings between two sets (like nodes in a graph), maps can store edges or connections, allowing for efficient traversal and lookups during graph algorithms.
- **Solving Anagram Problems:** Maps can help determine if two strings are anagrams by counting the frequency of each character in both strings and comparing the results. You can use the character counts as keys and their frequencies as values.
- **Dynamic Programming:** Maps are useful in dynamic programming scenarios where you need to store intermediate results keyed by specific states. For example, when solving the Longest Increasing Subsequence problem, a map can help store the lengths of subsequences based on ending values.