Welcome

Hello, and welcome to competitive programming. Today we will go over the C++ standard template library classes for stacks, queues, and lists.

Objectives

I expect you know the underlying data structures already, and instead want to talk about the options you have from the STL. These are simple enough you could just implement them yourself, but I would advise you to use the built-in library code if you can. After all, it's less coding for you this way, and one less source of a bug or inefficiency creeping in.

Stacks

The stack is one of the most used data-structures. Sometimes you will use one explicitly, but you are also using a stack implicitly whenever you use a recursive algorithm. Some of the more common easy problems involving stacks do things like check if braces or comments are matched. There are some graph algorithms like depth first search and strongly connected components that use stacks as well.

The built-in C++ stack library has the four major operations you would expect. Note that `pop` does not return an element; it is void. If you want to save the element you have to call `top` before calling `pop`.

Another interesting method is `emplace`. It creates a new instance of an object for you so you don't have to call `new` yourself.

Queues

Queues are everywhere also. You need a queue to implement breadth first search, and to simulate communication. If you look at the reference page for C++ queues, you'll see that you are allowed to specify the underlying container type. By default, C++ uses a deque (or doubly ended queue) for the container. This gives you the ability to access both ends of the queue in constant time. You should know that the queue class does not give you full access to both ends. You can read from the front and the back, but you can only push into the back and pop from the front.

Interestingly, they chose the names `push` and `pop` instead of `enqueue` and `dequeue`.

If you want complete doubly ended access, you will need to use the `deque` class explicitly.

Another word of warning: do not make assumptions about the underlying memory. In data structures, you were taught to use a list or an array, and with the array you could use pointer arithmetic to access elements. That won't work here, since in order to give you these access time guarantees the memory layout is more complicated.

Lists

If you insist, C++ does have a doubly linked list class. If you need a doubly linked list, you are definitely advised to use this class rather than making your own implementation. The need to keep `next` and `previous` pointers in sync is a rich source of bugs.

One other thing is that you almost never will want to use this structure. A vector will be faster in almost all circumstances, with one notable exception: if you must insert data in the middle of the structure, then a linked list can do that more efficiently than an array list.

That’s it

Well, that’s it for now. We will release a problem set, and I want you to focus on two goals: first, be able to triage quickly the kind of data structure you will need, and second, of course, is to make use of the C++ STL classes. There are links to the reference pages on the course web site to show you more of the capabilities of these classes.

Next
Transcript — TITLE →