

An Introduction to programming Through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture No. 24 Part- 3
Data structure based programming
Implementation of standard library data structures

Welcome back.

(Refer Slide Time: 0:22)



What we discussed

- Standard library class set
 - Many other operations
 - Iterators
- Standard library class unordered_set
- Standard library class pair
 - Useful for quick programming
 - Pairs used in maps: Map elements are pairs <keytype,value type>
 - Similar class "tuple" also provided
- Standard library has other interesting classes

Next: Remarks on how data structures are implemented



In the previous segment we discussed the Standard Library Class Set and Pair and also briefly talked about unordered sets. In this segment, we are going to discuss a little bit about how data structures are implemented. So, all these things sound like magic, but, of course, there is some code which is running behind them so I, want to tell you about that code.

(Refer Slide Time: 0:44)

Implementation of standard library data structures

Vector:

- Implementation similar to string class we discussed
- Key question:
 - what if push_backs requires more than currently allocated array?
 - Must allocate new array, copy content, delete current array.

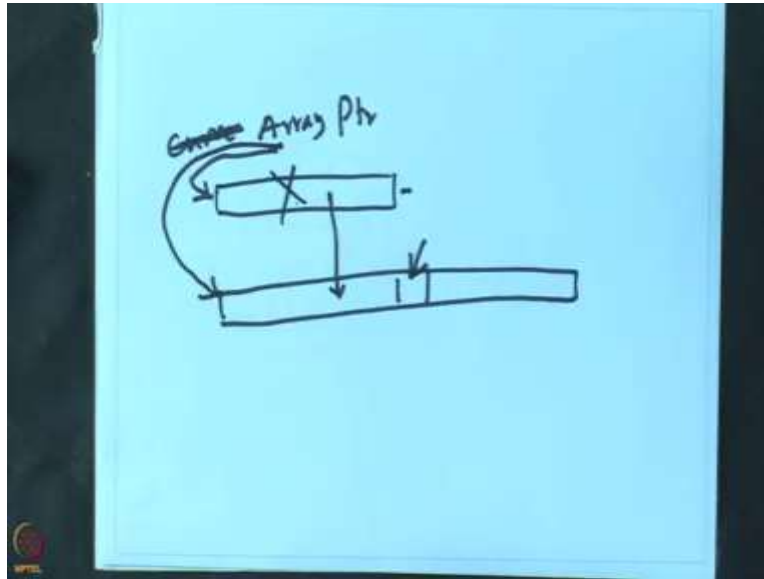


So, in the first class that I am going to discuss is the Vector Class. Now, the Vector class is implemented in a manner similar to the String Class I, should I guess capitalize this because by the String that I have, that I mean over here, I mean the string that we discussed in our lecture, not the Standard Library Class String.

If you remember, so we discussed a data structure or a class for storing character strings and we ended up writing things like constructors, assignment operators, copy constructor, destructor and also concatenation operators. So, the Vector Class has been implemented by someone in a similar manner. So, what is the difficulty, what is what is sort of the key difficulty in implementing a Vector Class, what is the key question we need to answer?

Well, if we are doing a push back operation and if the current array that has been allocated to hold the elements of the vector gets exhausted, what I mean by that is, currently you have 100 elements allocated and you push back one more element, so now you have 101 elements which are stored in your vector. What do you do? Well clearly if you remember what we did in the string case, what you will have to do is you will have to allocate a new Array. Then you copy the content of this array into that new array and then you delete the current array.

(Refer Slide Time: 2:27)



So, if I may draw a picture, this is your current array or array, this is the array pointer and this is pointing to this array. So if I do a push back I really want to store that element somewhere over here. But this area is not given to me, so what do I do, I ask for a bigger array, I may just ask for an additional element and I may just ask for just a little bit more, but it turns out it is good to ask for a much larger array.

So then I copy this part to this part, so I get whatever I had earlier into this array, and then I insert whatever I want to push back I store that element that I want to push back into this element over here. And then I set my array pointer to point to this new array and I delete this array. I give this array back to my storage allocator because I do not need it any longer. So, this is something I want to do, so that is sort of natural thing and that is, in fact, exactly what happens.

(Refer Slide Time: 3:52)

Implementation of standard library data structures

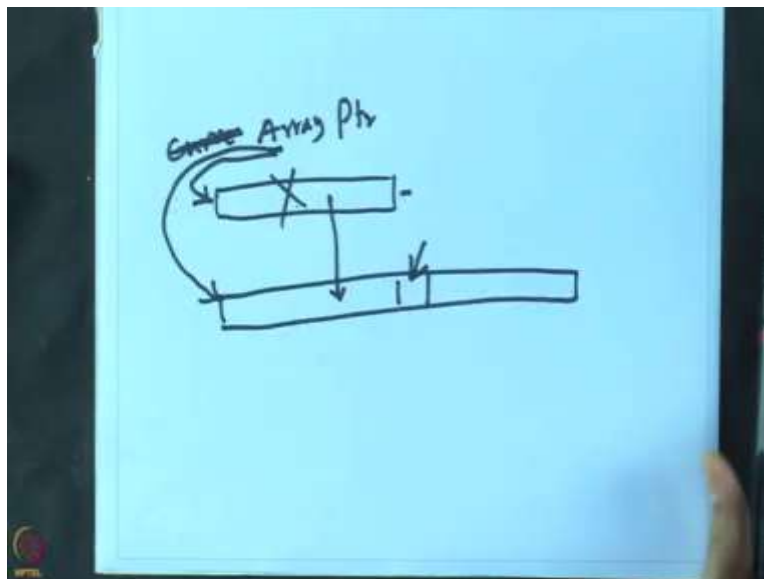
Vector:

- Implementation similar to string class we discussed
- Key question:
 - what if push_backs requires more than currently allocated array?
 - Must allocate new array, copy content, delete current array.
- How big an array to allocate?



But there is an interesting question that I have not answered which is how big an array do I allocate.

(Refer Slide Time: 4:01)

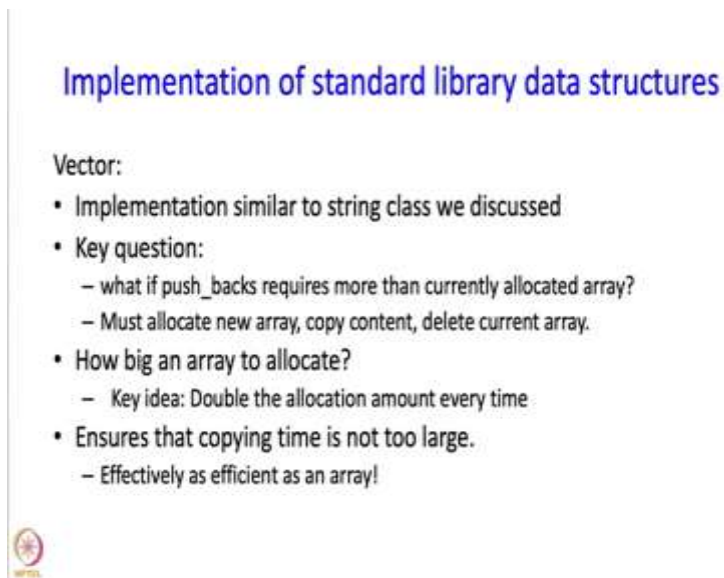


Now, turns out that it is a good idea to double the size. So, why is that, if I double the size. then subsequently if you do more push backs, you will not immediately have to

allocate and copy and things like that. So you will be saving on your copying cost and additional allocation cost.

You will be wasting a little bit about little bit of memory but how much memory are you wasting. Well you are wasting exactly as much memory potentially as you have elements, so you are wasting a little bit of memory, but the time you save because of not having to copy too frequently is a lot more important.


(Refer Slide Time: 4:47)



Implementation of standard library data structures

Vector:

- Implementation similar to string class we discussed
- Key question:
 - what if push_backs requires more than currently allocated array?
 - Must allocate new array, copy content, delete current array.
- How big an array to allocate?
 - Key idea: Double the allocation amount every time
- Ensures that copying time is not too large.
 - Effectively as efficient as an array!



And it turns out that it is possible to prove in fact, that if you sort of do this doubling idea then the copying time is not too large. In fact, you can then think of vectors as being as efficient as arrays. So, when you have an array, you do not have to do any copying, right. The memory is given to you once and for all. Here, you do have to do some copying once in a while you have to do allocate ask for new memory.

But what this is saying is that if you have this doubling strategy then you can ignore those costs. Let me acknowledge that I (have) I have not explained exactly why this happens. It can be explained as I said you can actually prove rigorously what I am saying but that is not a part of this course. You may see it in some other courses say a course on data structures. But it is worth knowing that vectors really are practically as efficient as arrays.

(Refer Slide Time: 6:07)

Implementation of (ordered) set

- Very tricky
- Elements of the set are “effectively” stored in sorted order.
- Deciding whether a certain element is present:
 - Happens using a binary search like idea
 - Time proportional to $\log(\text{number of elements present})$
- Maps are similar.
- Unordered sets and unordered maps are even cleverer!



Let me next, say a few words about implementation of ordered sets. First of all, let me acknowledge that this implementation is very tricky. We have not even begun to think about how ordered sets can be implemented in this course. The Ideas are quite-quite clever and we have not seen them. Having said that, let me just give you a hint. What happens when you store an ordered set?

Well, of course, there is some memory allocation that happens but at the end you get the effect that all your set elements are effectively stored in some sorted order. So they are not actually, they are not actually in sorted order, but for practical purposes they are in sorted order. Again this is intriguing and I am sorry to intrigue you but well may be this is this should be an inspiration for you to take the data structures course. Anyway, what happens is that effectively the elements get stored in a sorted order.

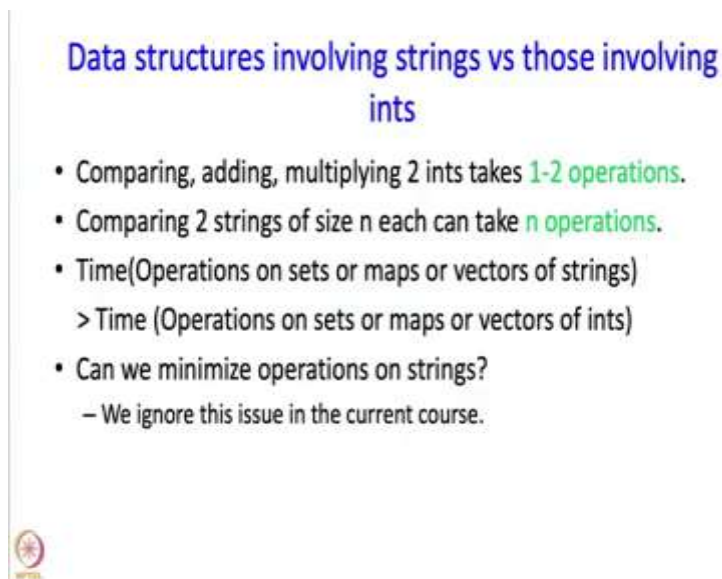
And why is sorted order important, well that you know the answer to. So if I want to decide whether an element is present, what can I do? Well I can do binary search so this binary search like idea also works and we have discussed binary search long ago and we have seen that binary search is actually very fast, you do not have to look at all the elements of the sets.

You only look at log of the size of the set that many elements. So the time proportion, the time required to determine the current element is present is proportional to the log of the number of elements. So this happens very fast and that is another reason for you to be using a set, the set class. Because it is actually a very it actually has very fast operations.

And let me just mention over here, that maps which you studied, which we studied in the last lecture are also implemented in the similar manner. So, again, if I want to store something into a map, if I write A of something equal to something, then that happens in time proportional to log or if I want to get the value stored in the map again that happens in time proportional to the log of the number of pairs stored in the map.

Now, again one very-very cryptic remark I should make and that is that unordered sets and unordered maps are even cleverer and they actually work even faster. I am not going to say anything about it, you will need to take course on data structures if you want to understand why unordered sets and unordered maps are faster and what is the clever idea behind them.

(Refer Slide Time: 9:15)



Data structures involving strings vs those involving ints

- Comparing, adding, multiplying 2 ints takes 1-2 operations.
- Comparing 2 strings of size n each can take n operations.
- Time(Operations on sets or maps or vectors of strings)
> Time (Operations on sets or maps or vectors of ints)
- Can we minimize operations on strings?
 - We ignore this issue in the current course.

I want to make one more comment again this relates to the efficiency of these classes or data structures. So, basically a point that I want to make, is that if I am defining a class involving strings, it is usually going to be more complex or more time consuming than a

class involving ints. So, why is this, well if I want to compare, if I want to do any operation on ints, our computers are very capable of operating on ints or in general on numbers. So, all these operations take one-two cycles on most computers. So they involve one two machine level instructions if you remember them. On the other hand, if I am comparing two strings say they are of size 'n' then potentially I will have to compare all characters, all the elements of the string and could take as large as 'n' operations. So, that is the basic issue that if I have things which are built up using strings, then strings are long obviously. And therefore, the time taken to do something with strings is usually longer than the time taken to do something with integers. So, in other words what I am saying is that if you are counting, if you are looking at the time for operations on sets or maps or vectors of strings, that is usually going to be bigger than the time for operations on sets or maps or vectors of ints and therefore, it raises the question can we minimize the operations on strings.

So, if you are writing production quality programs, this will be an important question you will want to minimize the operations that you perform on strings. This is a point that I want to raise right now just to alert you to it, but in this course, in this course we are not going to worry about this point. So, in this course we are going to ignore this issue but this is just a point that I want to want you to keep at the back of your mind and again you can think about it when you do the data structures course you will see why this is the case.

(Refer Slide Time: 11:41)



What we have discussed

- Implementation of vector and set
- Maps are implemented like set: log time needed for indexing

Next: Composing sets, vectors, maps, strings..



Alright, so what did we discuss in this segment? Well, we discussed implementation of vector and set and we said that maps are implemented like sets and the beauty of maps is that for insertions as well as for search, for determining whether something is present in the map, logarithmic time is needed. Next I am going to talk a little bit about how to compose these data structures together. So, how do you build sets of vectors? When do you build sets of vectors or vectors of maps or maps of strings and so on? But again, before that let us take a quick break.