CSCI 136: Data Structures and Advanced Programming
Lecture 20
Ordered Structures
Instructor: Dan Barowy
Williams

Topics

• Ordered structures

Your to-dos

1. Lab 6 (partner lab), due Tuesday 4/12 by 10pm.
2. Read before Wed: Bailey, Ch 12-12.5.

Announcements

  Organized by Prof. Kelly Shaw and CoSSAC.
  Wednesday, April 13 from 7-7:45pm in TBL 211.
  “Extra special snacks” provided by CoSSAC afterward in the Eco Cafe.

• This Friday’s colloquium: CS pre-registration info session.
Announcements

Please consider being a TA next semester (especially for this class!)

Applications due Friday, April 22.

https://csci.williams.edu/tatutor-application/

Ordered structures

structure5 Stack implementations

A structure is an interface for a “traversable” collection of objects. In other words, it represents a class that contains some number of elements, and those elements can be iterated, added, and removed. Membership and size can also be checked.

Most of the data structures we discuss in this class implement structure.
public interface Structure<E> extends Iterable<E> {
    public int size();
    public boolean isEmpty();
    public void clear();
    public boolean contains(E value);
    public void add(E value);
    public E remove(E value);
    public java.util.Enumeration elements();
    public Iterator<E> iterator();
    public Collection<E> values();
}

Why is a `Structure` interface a good idea? What benefit do we get from having it?

One reason

Suppose we write a method that takes a `Structure`. We could give it an instance of any data structure that implements the `structure` interface.

E.g., we could iterate over the elements and print them because all structures have the `iterator()` method.

What about order?

Does the `Structure` interface require that elements be ordered?
### structure in structure5

```java
public interface Structure<E> extends Iterable<E> {
    public int size();
    public boolean isEmpty();
    public void clear();
    public boolean contains(E value);
    public void add(E value);
    public E remove(E value);
    public java.util.Enumeration elements();
    public Iterator<E> iterator();
    public Collection<E> values();
}
```

### What about order?

Does the `structure` interface require that elements be **ordered**?

No.

Is order a property that **could be enforced** using interfaces?

No. Order is a **data-dependent property**, so there’s no way to check whether something is ordered until runtime.

### OrderedStructure

Nonetheless, we can **signal our intent** with an interface.

**How would we write an `OrderedStructure` interface?**

Do its elements need to have **any special property**? (i.e., how would we **compare** them?)

Let’s try to write this.

```java
OrderedVector
```

Let’s try implementing an `OrderedVector`.  

```java
(OrderedVector)
```
OrderedVector

How do we figure out where `add` should insert?

Binary search to the rescue.

Binary search

<table>
<thead>
<tr>
<th>100</th>
<th>101</th>
<th>322</th>
<th>365</th>
<th>423</th>
<th>478</th>
<th>499</th>
<th>504</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Want to know whether the array contains the value 322, and if so, what its index is.

Binary search is a divide-and-conquer algorithm that solves this problem.

Binary search is fast: in the worst case, it returns an answer in $O(\log_2 n)$ steps.

Important precondition: array must be sorted.

Binary search

Looking for the value 322.
Binary search

Looking for the value **322**.

322 = 365? no
322 < 365? yes
Binary search

Looking for the value 322.

322 = 101? no
322 < 101? no
322 > 101? yes

322 = 322? yes
return 2
Recap & Next Class

**Today:**

Ordered structures

**Next class:**

More iterators
Bitwise operations