Stacks, queues, sets and maps

Dr Tom Ridge

2018 Q1

Stacks and queues

Overview

- The following slides introduce essential datastructures, stacks and queues
- These are closely related to lists. The difference is in which operations are supported.
- For example, stacks only allow you to add an element on the front of a list, and remove an element from the front of a list.

Stacks

Stacks are, essentially, lists.

Stack API, intro

Push puts an object on the top of the stack.
Pop takes an object from the top of the stack.
Stacks are LIFO: Last In First Out

Typical stack usage

```
s = new Stack();

s.push(1); s.push(2); s.push(3);
s;
s.pop(); // returns 3, s is changed to [1,2]
// note that [1,2] is Java's way of printing a stack with 2
s;
s.pop(); // returns 2, s is changed to [1]
s;
s.push(4);
s;
s.push(5);
s;
s.pop(); // returns 5
s;
```

Stack API, with state

We have used lists that are "immutable" - whenever you call `hd`, `tl`, etc you get back a new list, and the old list is not changed.
This makes things very easy to understand.
Most Java programs operate with objects that do have state. The state is "mutable" - you can change it.
The following is an API for stacks, with state.

Stack API

```
class MyStack { // N.B. Java already has a stack
                // class- this is just for us

  public void push(Object o) { ... }

  public Object pop() { ... }

  public boolean isEmpty() { ... }

  public String toString() { ... }

}
```

Stack implementation

```
class MyStack {
  // a stack is like a list, but cons is called
  // push, and hd is called pop, and the only other
  // method is isEmpty

  private List l = nil();

  public void push(Object o) { l = cons(o,l); }

  public Object pop() {
    Object h = hd(l); l = tl(l); return h; }

  public boolean isEmpty() { return l.isEmpty(); }

  public String toString() { return l.toString(); }

  // pretend list operations hd, tl etc go here
```

Questions on stacks (and queues)

See module webpage

Queues

Queues are, essentially, lists.

Queue API, with state

Note that enqueue adds an object "at the end" of the queue.
Dequeue removes an object from the front of the queue.
N.B. Queues are FIFO: First In First Out

```
class MyQueue {
  // N.B. Java already has a queue class- this is just
  // for us

  // called "offer" in java.util.Queue, or "add" in
  // java.util.Collection
  public void enqueue(Object o) { ... }

  // changes the current list! use with care
  public Object dequeue() { ... }

  public boolean isEmpty() { ... }

  public String toString() { ... }
}
```

Queue implementation

```
class MyQueue {
  // a queue is like a list, but you add elements at
  // the end (you still remove elements from the front)
  private List l = nil();

  // called "offer" in java.util.Queue, or "add" in
  // java.util.Collection
  public void enqueue(Object o) { l = append1(l,o); }

  // changes the current list! use with care
  public Object dequeue() {
    Object h = hd(l); l = tl(l); return h; }

  public boolean isEmpty() { return l.isEmpty(); }

  public String toString() { return l.toString(); }
  // pretend list operations hd,tl etc go here!
}
```

Sets and maps

Sets

- Recall the notion of a set from previous modules.
- A set is a collection of things.
- Given a thing, and a set, you can check whether the thing is in the set, or not. Unlike a list you **cannot** find out how many of a given thing is in the set.
- Sets are absolutely fundamental to mathematics, and also computer science.
- It is important to distinguish the mathematical notion of a set, from the computer science notion. They are related, but usually not the same.

Set notation

The set containing the integers 1, 2 and 3: $\{1,2,3\}$
Numbers from 1 to 1000: $\{1, 2, 3, \ldots, 1000\}$

Some subtleties about sets

- The sets $\{1,2,3\}$ and $\{3,2,1\}$ are the same set (order does not matter).
- The sets $\{1,2,3\}$ and $\{3,3,2,2,2,1,1,1,1\}$ are the same set (there is no notion of "duplicate" elements).
- Finite sets have a size. The size of $\{1,2,3\}$ is 3.
- The size of $\{3,3,2,2,2,1,1,1,1\}$ is ...

Consequences of equality

- If x=y, then f(x) = f(y)
- So, if s1=s2, then size(s1) = size (s2)
- $\{1,2,3\} = \{3,3,2,2,2,1,1,1,1\}\ldots$
- ... so size($\{1,2,3\}$) = size($\{3,3,2,2,2,1,1,1,1\}$) = 3

Sets in Java

See the Java API documentation, java.util.Set

```
s = new HashSet();
s.add(1);
s.add(2);
s.add(3);
s.size(); // returns 3
s.add(3);
s.size(); // returns 3
s.remove(3);
s.size(); // returns 2
s.contains(1); // true
s.contains(4); // false
```

What use are sets?

Suppose we have an array of marks. How to quickly test whether a given mark is in array or not?

```
marks = Arrays.asList(8,9,10,1,3,5);
s = new HashSet(marks);
s.contains(2);
s.contains(3);
```

The point is: using `contains()` on an object (that is a Set) is much much much more efficient than looking through the array each time.

Other things you can do with sets: set difference

Set difference: {1,2,3} \ {2,4} = {1,3}

```
s1 = new HashSet(Arrays.asList(1,2,3));
s2 = new HashSet(Arrays.asList(2,4));
s1.removeAll(s2);
s1.contains(1);
s1.contains(2);
```

Set union

Set union: {1,2,3} union {2,4} = {1,2,3,4}

```
s1 = new HashSet(Arrays.asList(1,2,3));
s2 = new HashSet(Arrays.asList(2,4));
s1.addAll(s2);
s1.contains(4);
s1.contains(5);
```

Maps

- Mathematically, functions are sets of pairs.
- In computer science, we tend to think of functions as e.g. methods on a class. A function is closely tied to the code that implements the function.
- For example, the following is a function:

```
int double(int x) { return 2*x; }
```

- Mathematically, one might think of the set of pairs $\{(1,2),(2,4),(3,6),\ldots\}$
- Here the first element in each pair is the "input", and the second is the "output".

Maps

Maps are another model of functions. Maps must be finite (so you wouldn't really want to represent the double function as a map). See Java API documentation, `java.util.Map`

Basic map operations

The basic map operations are:

- put, which allows you to put a pair of a key (the input) and a value (the output) into the map.
- get, which takes a key and returns the corresponding value in the map (if any)

See Java API documentation, `java.util.Map`

Maps in Java

```
m = new HashMap();
m.put(1,2);
m.put(2,4);
m.put(3,6);
m.get(2);
m.get(4);
m.put(3,"hello");  // override previous value
m.get(3);
```

Examples of using a map

Suppose we have a set of people, and their ages. Given a person, how to get their age?

```
m = new HashMap();
m.put("alf",12);
m.put("bert",23);
m.put("charlie",34);
m.get("alf");  // returns 12
m.get("charlie"); // returns 34
```

My advice

- One of the first steps in becoming a programmer is to understand sets and maps, and use them in appropriate situations.
- Almost all of the code I write uses lists, sets and maps.
- Do yourself a favour and learn what they are and how they are used. Learn what methods are supported by the Java implementations.

Example questions

Given a list of integers:

- find the first duplicated element
- find all duplicated elements
- find the number of times each element appears

Java collections

- Lists, sets, maps, queues, etc are collections of things. Java collections are standard classes for dealing with these collections.
- http://docs.oracle.com/javase/tutorial/collections/ - Java collections tutorial
- http://docs.oracle.com/javase/tutorial/collections/interfaces - Java collections core interfaces
- Individual classes such as HashSet - http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html