122COM: Data structures

David Croft

November 25, 2015

Abstract

This week we are going to be looking at different data structures and how to implement your own Abstract Data Types.

1 Introduction

As you write more and more advanced software you are going to encounter different data structures and types. Data structures and types are how we store and organised multiple pieces of information in our software.

This week you will be writing your own implementations of some of the more common data types. Unless otherwise stated you will be writing array based implementations, there is an option to explore linked list implementations as part of the advanced tasks.

The source code files for this lab and the lecture are available at:

```
https://github.com/dscroft/122COM_data_structures.git
```

```
git@github.com:dscroft/122COM_data_structures.git
```

2 Stacks

A stack is a Last In First Out (LIFO) linear data type that can be imagined as being like a pile of paperwork. New elements can only be added to the top of the stack. Old elements can only be removed from the top of the stack. Elements at the bottom or in the middle of the stack cannot be reached except by removing all the elements above them.

Stacks very simple data structures as elements can only be added/removed from one end. Adding elements to a stack is normally referred to as pushing that element onto the stack. Removing an element is traditionally referred to as popping an element from the stack. Table 1 on the following page shows the effect that popping and pushing has on an example stack.

	0	1	2	3	
bottom →	A	B	C	D	← top
	A	B	C		pop()
	A	B			pop()
	A	B	Z		push(Z)

Table 1: Stack behaviour example.

Pre-lab work: Here's one I made earlier

Although C++ has a `stack` type already included in and the Python `list` can behave like one it's going to be useful for you to understand how to create your own data structures.

1. Implement a stack using an array.
   - You can use pre_stack as a starting point, it comes complete with the basic structure and a series of tests to check that your stack is working as expected.
   - You will need to complete the functions to pass all the tests.

2.1 Size limitations

In your stack code the contents of the stack were held in an array. The pop() and push() functions were used to modify that array in the ways allowed by a stack. Using an array has the advantage of being very simple to program but has a major disadvantage of limited the maximum number of elements that we can store in the stack as an array has a predefined size. If we try to store more elements than the array can hold then the stack can't do it.

There are three possible solutions:

1. Create our stack with the largest capacity we could ever need. This unfortunately means wasting a huge amount of memory if we don't end up needing all that capacity.
2. Change our stack/queue so that if the internal array runs out of space a new, bigger array is created. This unfortunately means spending a lot of time copying the contents of one array into another which is slow.
3. Use linked lists instead of arrays.

Extended work: Linked list stack

2. Try and implement a stack using a linked list. You may find it helpful to use your array implementation as a starting point.
   - If you are using C++ then you will need to use pointers and dynamic memory allocation.

3 Queues

A First In First Out (FIFO) structure. Often used as a buffer to hold items for processing in the order in which they arrive. New elements can be added to the back of queue only. Old elements can be removed from the front of the queue only. There is no access to the middle of the queue.

C++ and Python both have built in queue types. The default Python queue is actually a dequeue (double ended queue) which allows you to push and pop from both ends of the queue but it can still be used as a normal queue.

Lab work: Queues

3. Complete the supplied lab_queue code to implement your own queue type.
   - You many find it helpful to refer back to your stack code.
   - Test that it works correctly.
4. Is it better to use an array or a linked list to implement a queue? Explain your answer.

3.1 Priority Queues

A variation on the traditional queues. Same element access rules as a normal queue, but all the elements have an associated priority. Elements within the queue are sorted based on their priority. Elements with the same priorities maintain their ordering.

Priority queues can be thought of as being like Hospital ER queues. Serious injuries (high priority) are seen first. Otherwise everyone is seen in the same order that they arrived in.

	0	1	2	3	
front →	A	B	C	D	← back
	B	C	D		pop()
	C	D			pop()
	C	D	Z		push(Z)

Table 2: Queue behaviour example.

Extended work: Priority queues

5. Research the behaviour of priority queues.
6. Using your existing queue code as a starting point, implement a priority queue.
    - When elements are added to the queue they will need to supply a priority.
    - You will need to consider how this priority affects where in the queue new elements get placed.

4 Sets

Sets are unordered collections of unique elements. Or to put it another way, a set cannot have more than one element with the same value. If multiple elements with the same value are added to a set then only one copy is kept. Elements can be removed from anywhere in the set.

As with queues and stacks there are multiple ways to actually implement a set. The simplest way would be to use an array and just iterate over the contents of the array every time a new element is added to make sure that it was not already present. This would be easy to code but the resulting set would be limited by the size of the array and inefficient as every value would have to be checked to prevent duplicates, an $O(n)$ insertion.

The size issue could be solved using a linked list but we would still need to check every value.

A better choice would be to use a Binary Search Tree (BST). As we saw in the lecture section, it is very fast to search a BST for a given value. As long as we keep our BST balanced, a set implemented using a BST would be very fast.

Both Python and C++ have built in set types called sets. Python sets have an advantage that they offer built-in functions which allow you to perform mathematical set operations such as intersection, union and difference.

Lab work: Queues

7. Using lab_set as your starting point implement your own set type using an array.
    - You may find it helpful to refer back to your stack and queue code.
    - Test that your set works correctly.