Lists4 Study Guide | CS 61B Spring 2018

sp18.datastructur.es/materials/lectures/lec6/lec6

Lecture Code

Code from this lecture available at

https://github.com/Berkeley-CS61B/lectureCode-sp18/tree/master/lists4.

Overview

Lists vs. Arrays Our `DLList` has a drawback. Getting the ith item is slow; we have to scan through each item in the list, starting from the beginning or the end, until we reach the ith item. For an array named `A`, however, we can quickly access the ith item using bracket notation, `A[i]`. Thus, our goal is to implement a list with an array.

AList The `AList` will have the same API has our `DLList`, meaning it will have the same methods as `DLList` (`addLast()`, `getLast()`, `removeLast()`, and `get(int i)`). The `AList` will also have a `size` variable that tracks its size.

AList Invariants There are a few invariants for our `AList`.

- `addLast`: The next item we want to add, will go into position `size`.
- `getLast`: The item we want to return is in position `size - 1`.
- `size`: The number of items in the list should be `size`.

Implementing AList Each `AList` has an `int[]` called `items`.

- For `addLast`, we place our item in `items[size]`.
- For `getLast`, we simply return `items[size - 1]`.
- For `removeLast`, we simply decrement `size` (we don't need to change `items`). Thus, if `addLast` is called next, it simply overwrites the old value, because size was decremented. **However, it is good practice to null out objects when they are removed, as this will save memory.** Notice how closely these methods were related to the invariants.

Abstraction One key idea of this course is that the implementation details can be hidden away from the users. For example, a user may want to use a list, but we, as implementers, can give them any implementation of a list, as long as it meets their specifications. A user should have no knowledge of the inner workings of our list.

Array Resizing When the array gets too full, we can resize the array. However, we have learned that array size cannot change. The solution is, instead, to create a new array of a larger size, then copy our old array values to the new array. Now, we have all of our old values, but we have more space to add items.

Resizing Speed In the lecture video, we started off resizing the array by one more each time we hit our array size limit. This turns out to be extremely slow, because copying the array over to the new array means we have to perform the copy operation for each item. The worst part is, since we only resized by one extra box, if we choose to add another item, we have to do this again each time we add to the array.

Improving Resize Performance Instead of adding by an extra box, we can instead create a new array with `size * FACTOR` items, where `FACTOR` could be any number, like 2 for example. We will discuss why this is fast later in the course.

Downsizing Array Size What happens if we have a 1 million length array, but we remove 990,000 elements of the array? Well, similarly, we can downsize our array by creating an array of half the size, if we reach 250,000 elements, for example. Again, we will discuss this more rigorously later in the course.

Aside: Breaking Code Up Sometimes, we write large methods that do multiple things. A better way is to break our large methods up into many smaller methods. One advantage of this is that we can test our code in parts.

Generic AList Last time, we discussed how to make a generic `DLList`. We can do something similar for `AList`. But we find that we error out on array creation. Our problem is that generic arrays are not allowed in Java. Instead, we will change the line:

```
items = new Item[100];
```

to:

```
items = (Item[]) new Object[100];
```

This is called a cast, and we will learn about it in the future.

Exercises

C Level

1. Complete the exercises from the online textbook here.

B Level

1. We did not touch upon the method `addFirst` for an `AList`. Think of some of the problems you would experience in writing an `addFirst` method, and think of some potential solutions. If you think you've got a good one, write it out.

2. Do the bonus question from this slide