CSE 250: ADTs, Sequences, and Arrays

Lecture 8

Sept 15, 2023

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Reminders

- PA1 Tests due Sun, Sept 17 at 11:59 PM
  - Recitations will cover writing good test cases.
- PA1 Implementation due Sun, Sept 24 at 11:59 PM
  - Implement a Sorted Linked List
- Autolab etiquette: Test locally *first*.
  - If you (repeatedly) submit code that doesn't compile to autolab, we reserve the right to mock you mercilessly.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Sequences

- 1, 2, 3, 4, 5
- 4, 7, 2, 13, 8, 12, 14, 20, 12, 1
- 17, 14, 20, 14, 17, 12, 8, 9, 8, 6
- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . . (Fibbonacci Sequence)
- 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'
- The lines of a file

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

What makes a sequence

What is common to these?

- A collection of elements of some type (e.g., numbers, characters, strings).
  - Two elements can have the same value.
- Each element has a unique position.
  - No two elements have the same position
- The positions of all elements are contiguous.
  - For an N element sequence, every position from $[0, N)$ is occupied.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

What can we do with a sequence?

What are the simplest things we can do with a sequence?

- Get the 'i'th element.
- Modify the 'i'th element.
- Enumerate the elements in order.
- Get the number of elements

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Abstract Data Types

Set the 'i'th element

Get the 'i'th element

Count the elements

Enumerate the elements

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Abstract Data Types

Abstract Data Type defines...

- Domain: What kind of data is stored? (e.g., elements, key/value pairs)
- Constraints: How are items related? (e.g., ordered keys)
- Operations: How can the data be accessed/modified (e.g., 'i'th item)

Like a Java `interface`[1]

[1] The term `interface` is not quite the same as ADT; The interface only formalizes the permitted operations.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

The Sequence ADT

```
public interface Sequence<E>
{
  public E get(int idx);
  public void set(int idx, E value);
  public int size();
  public Iterator<E> iterator();
}
```

E is the type of thing in the Sequence.

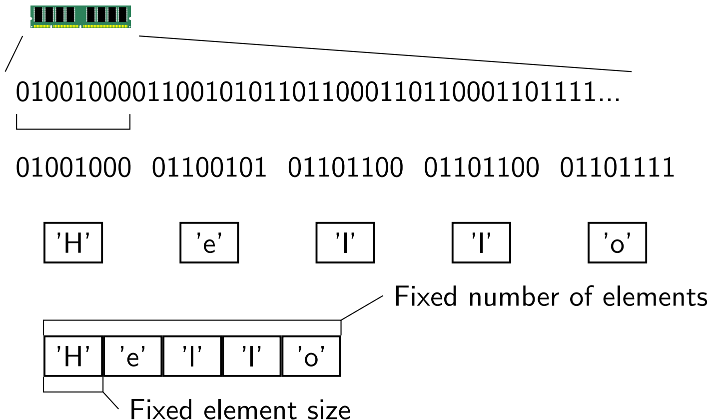
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

The Sequence ADT

"The Wizard of Oz"; ©1939 Metro-Goldwyn-Mayer

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

CSE 220 Crossover



OpenClipArt: https://freesvg.org/random-access-computer-memory-ram-vector-image

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

RAM

`new T` finds some unused part of memory big enough to fit a `T`, marks it used, and returns the *address*[2].

```
1    int[] data = new int[50];
```

... allocates $4 \cdot 50 = 200$ bytes[3].

If `data` is at address a: where do you look for `data[19]`?
$a + 4 \cdot 19 = a + 76$

How long does it take to locate `data[19]`?
$\theta(1)$

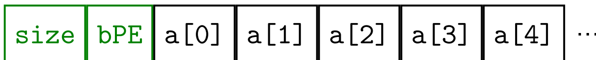
[2] You'll get *very* familiar with memory allocation in 220.
[3] An `int` in java is 4 bytes (32 bits).

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Arrays

What information goes into an `T[]` array?

- `size`: 4 bytes for the number of elements[4].
- `bytesPerElement`: 4 bytes for `sizeof(T)`[5].
- `data`: `size` $\times$ `bytesPerElement` bytes.



[4] Some languages (e.g., C) skip this, relying on the programmer to track it.

[5] Some languages (e.g., C, C++) skip this, since it's fixed at compile time.

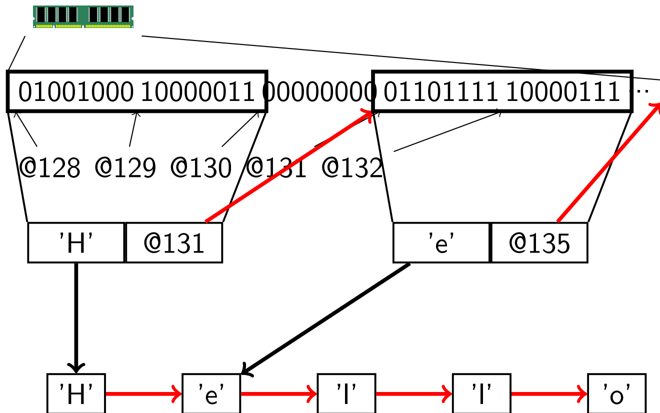
© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

How do we implement...

- `public E get(int idx)`
  - Return bytes $\texttt{bPE} \times \mathit{idx}$ to $\texttt{bPE} \times (\mathit{idx} + 1) - 1$
  - $\theta(1)$ (if we treat `bPE` as a constant)
- `public void set(int idx, E value)`
  - Update bytes $\texttt{bPE} \times \mathit{idx}$ to $\texttt{bPE} \times (\mathit{idx} + 1) - 1$
  - $\theta(1)$ (if we treat `bPE` as a constant)
- `public int size()`
  - Return `size`
  - $\theta(1)$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

CSE 220 Crossover 2: List Harder



OpenClipArt: https://freesvg.org/random-access-computer-memory-ram-vector-image

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Linked Lists

```
1    public class LinkedListNode<T>
2    {
3      T value;
4      LinkedListNode<T> next = null;
5    }
```

```
1    public class LinkedList<T> implements List<T>
2    {
3      LinkedListNode<T> head = null;
4      /* ... */
5    }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

How do we implement...

- `public E get(int idx)`
  - Start at `head`, and move to the `next` element `idx` times. Return the element's `value`.
  - $\theta(idx)$, $O(N)$
- `public void set(int idx, E value)`
  - Start at `head`, and move to the `next` element `idx` times. Update the element's `value`.
  - $\theta(idx)$, $O(N)$
- `public int size()`
  - Start at `head`, and move to the `next` element until you reach the end. Return the number of steps taken.
  - $\theta(N)$

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Linked Lists' `size`

Can we do better?

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Store size

```
1    public class LinkedList<T> implements List<T>
2    {
3      LinkedListNode<T> head = null;
4      int size = 0;
5      /* ... */
6    }
```

- How expensive is `public int size()` now?
  ($\theta(1)$)
- How expensive is it to *maintain* `size`?
  (Extra $\theta(1)$ work on insert/remove).

Storing redundant information can reduce complexity.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Optional

```
1 public class LinkedListNode<T>
2 {
3   T value;
4   LinkedListNode<T> next = null;
5 }
```

```
1 public class LinkedList<T> implements List<T>
2 {
3   LinkedListNode<T> head = null;
4   /* ... */
5 }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

null is risky

```
1  public T get(int idx)
2  {
3    LinkedListNode<T> curr = head;
4    for(int j = 0; j < i; j++)
5    {
6      curr = curr.next;
7    }
8    return curr.value;
9  }
```

NullPointerException

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

null is risky

```
1  public T get(int idx)
2  {
3    LinkedListNode<T> curr = head;
4    for(int j = 0; j < i; j++)
5    {
6      if(curr.next == null){ /* do something useful */ }
7      curr = curr.next;
8    }
9    return curr.value;
10 }
```

It can be hard to remember when a value might be null.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Optional

```
1    public class LinkedListNode<T>
2    {
3      T value;
4      Optional<LinkedListNode<T>> next = Optional.empty();
5    }
```

```
1    public class LinkedList<T> implements List<T>
2    {
3      Optional<LinkedListNode<T>> head = Optional.empty();
4      /* ... */
5    }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Optional

```
1  public T get(int idx)
2  {
3    Optional<LinkedListNode<T>> curr = head;
4    for(int j = 0; j < i; j++)
5    {
6      curr = curr.next; ←
7    }
8    return curr.value; ←
9  }
```

Compiler Error! (No such method)

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Optional

```
1  public T get(int idx)
2  {
3    Optional<LinkedListNode<T>> curr = head;
4    for(int j = 0; j < i; j++)
5    {
6      curr = curr.get().next; ←
7    }
8    return curr.get().value; ←
9  }
```

Compiler Error! (Unhandled Exception)

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Optional

```
1  public T get(int idx)
2  {
3    Optional<LinkedListNode<T>> curr = head;
4    for(int j = 0; j < i; j++)
5    {
6      try {
7        curr.get().next;
8      } catch(NoSuchElementException e) {
9        /* do something useful */
10     }
11   }
12   try {
13     return curr.get().value;
14   } catch(NoSuchElementException e) {
15     /* do something useful */
16   }
17 }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Optional

Creating an Optional

- `Optional.empty()`: Like `null`.
- `Optional.of(x)`: Get an Optional with value `x`.
- `Optional.ofNullable(x)`: Like `.of(x)`, but return `.empty()` if `x == null`;

Using an Optional

- `.isPresent()`: Return true if a value is present.
- `.get()`: Return the value, or throw exception if not present.
- `.orElse(y)`: Return the value if present, or `y` if not.

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Referential Access

What's the complexity of getting the value of the i'th element of a `LinkedList`?

If you have a pointer to the i'th `LinkedListNode` of a `LinkedList`, what's the complexity of getting its value?

If you have a pointer to the i'th `LinkedListNode` of a `LinkedList`, what's the complexity of getting the value of the $i+1$'th element?

If you have a pointer to the i'th `LinkedListNode` of a `LinkedList`, what's the complexity of getting the value of the $i-1$'th element?

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Doubly Linked Lists

```
1    public class LinkedListNode<T>
2    {
3      T value;
4      Optional<LinkedListNode<T>> next = Optional.empty();
5      Optional<LinkedListNode<T>> prev = Optional.empty();
6    }
```

```
1    public class LinkedList<T> implements List<T>
2    {
3      Optional<LinkedListNode<T>> head = Optional.empty();
4      Optional<LinkedListNode<T>> tail = Optional.empty();
5      /* ... */
6    }
```

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY

Linked Lists

Next time...

- Referential Access
- The full `List` type (insertions/deletions).
- More on Doubly-Linked Lists

© 2023 Oliver Kennedy, Eric Mikida, The University at Buffalo, SUNY