Linked list visualized

[1, 8, 4, 7]

head

Look at code

Iterating through a linked list

Use a variable starting at the head and move it down the list

How will we know when we’re at the end?
Iterating through a linked list: printing it

```
Node finger = head;
while (finger != null) {
    System.out.println(finger.value());
    finger = finger.next();
}
```
Iterating through a linked list: printing it

head

---

Node finger = head;
while (finger != null) {
    System.out.println(finger.value);
    finger = finger.next;
}
Iterating through a linked list: printing it

```java
Node finger = head;
while (finger != null) {
    System.out.println(finger.data);
    finger = finger.next;
}
```

Iterating through a linked list: printing it

```java
Node finger = head;
while (finger != null) {
    System.out.println(finger.data);
    finger = finger.next;
}
```
Iterating through a linked list: printing it

```
head
  ↓
data: 1
  ↓next:
data: 8
  ↓next:
data: 4
  ↓next:
data: 7
  ↓next: null
```

Checking if a value is in the list

```
head
  ↓
data: 1
  ↓next:
data: 8
  ↓next:
data: 4
  ↓next:
data: 7
  ↓next: null
```

```
public boolean contains(int value){
    Node finger = head;
    while (finger != null) {
        System.out.print(finger.value());
        finger = finger.next;
    }
    return false;
}
```

Adding to the end

```
[1, 8, 4, 7]
```

```
head
  ↓
data: 1
  ↓next:
data: 8
  ↓next:
data: 4
  ↓next:
data: 7
  ↓next: null
```

```
Adding to the end: add 9 to the end

[1, 8, 4, 7]
```

```
head
  ↓
data: 1
  ↓next:
data: 8
  ↓next:
data: 4
  ↓next:
data: 7
  ↓next: null
```

```
find the end of the list
```

```
How do we know we're at the end?
```

```
```
How can we add a value to the end of the list?
Adding to the end: add 9 to the end

```java
Node finger = head;
while (finger.next() != null) {
    finger = finger.next();
}
finger.setNext(new Node(9));
```
Adding to the end

[1, 8, 4, 7]

```java
data 1
next:

Node finger = head;
while (finger.next != null) {
    finger = finger.next;
}
finger.setNext(new Node(1));
```

Any special cases we need to handle?

Adding to the end

[1, 8, 4, 7]

```java
head: null

Node finger = head;
while (finger.next != null) {
    finger = finger.next;
}
finger.setNext(new Node(1));
```

What would happen here?

Removing a value

[1, 8, 4, 7]

```java
public void addLast(int value) {
    if (head == null) {
        head = new Node(value);
    } else {
        Node finger = head;
        while (finger.next != null) {
            finger = finger.next;
        }
        finger.setNext(new Node(value));
    }
}
```

How can we remove a value from the list?
Removing a value: remove 4

[1, 8, 4, 7]

Find it

Removing a value: remove 4

[1, 8, 4, 7]

Need the node before it too!

Removing a value: remove 4

[1, 8, 4, 7]

splice it out

Removing a value: remove 4

[1, 8, 4, 7]

splice it out
Removing a value: remove 4

[1, 8, 4, 7]

Removing a value

Node finger = head.next;
Node prev = head;
while (finger != null && finger.value.equals(value)){
    prev = finger;
    finger = finger.next;
}
if (finger != null){
    prev.setNext(finger.next);
}
Removing a value

```
Node finger = head.next;
Node prev = head;

while (finger != null && finger.value.equals(value)){
  prev = finger;
  finger = finger.next;
}

if (finger == null) {
  prev.setNext(finger.next);
}
```

Removing a value

```
Node finger = head.next;
Node prev = head;

while (finger != null && finger.value.equals(value)){
  prev = finger;
  finger = finger.next;
}

if (finger == null) {
  prev.setNext(finger.next);
}
```

Removing a value

```
Node finger = head.next;
Node prev = head;

while (finger != null && finger.value.equals(value)){
  prev = finger;
  finger = finger.next;
}

if (finger == null) {
  prev.setNext(finger.next);
}
```

Removing a value

```
Node finger = head.next;
Node prev = head;

while (finger != null && finger.value.equals(value)){
  prev = finger;
  finger = finger.next;
}

if (finger == null) {
  prev.setNext(finger.next);
}
```
Removing a value

```
Node finger = head.next;
Node prev = head;

while (finger != null && !finger.value.equals(value)) {
    prev = finger;
    finger = finger.next;
}

if (finger != null) {
    prev.setNext(finger.next);
}
```

What would happen here?

```
Node finger = head.next;
Node prev = head;

while (finger != null && !finger.value.equals(value)) {
    prev = finger;
    finger = finger.next;
}

if (finger != null) {
    prev.setNext(finger.next);
}
```

Any special cases we need to handle?
Removing a value

Delete 1. Does it work?

Removing a value

Delete 1. Does it work?

Removing a value

Delete 1. Does it work?
Linked lists: fast or slow?

- add to the end: slow
- add to the front: fast
- contains: slow
- get: slow
- insert at an index: slow
- remove an element: slow
- remove from the front: fast
- set the value of an existing element: slow
- size: fast

Can we make any of these faster?

Linked list visualized

[1, 8, 4, 7]

head

How does this help us?
Linked list visualized

We don’t have to search for the end of the list!

tail.next = new Node(value);
tail = tail.next;

Any downsides?

Circularly linked list visualized

Where is head?
Circularly linked list visualized

[1, 8, 4, 7]

Linked lists: fast or slow?

- add to the end: fast
- add to the front: fast
- contains: slow
- get: slow
- insert at an index: slow
- remove an element: slow
- remove from the front: fast
- set the value of an existing element: slow
- size: fast

Is removing from the end fast?

No! Need access to previous
Doubly linked list visualized

Keep track of both the next and previous node in the list

head

prev: null

next:

prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

Is removing from the end fast?

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail

Doubly linked list visualized

head

next:

prev: prev:

data: 1

next: next:

prev:

data: 8

next: next:

prev:

data: 4

next: next:

prev:

data: 7

next: next:

prev:

null

tail
Doubly linked list visualized

Could I remove a node if I had a reference to it?

Doubly linked list visualized

Could I remove a node if I had a reference to it?
Doubly linked list visualized

head

prev: null

next:

prev:

data: 1

next:

prev:

data: 8

data: 4

data: 7

prev:

next:

How does this help us? Still have to find the node?

How does this help us? Still have to find the node?
Doubly linked list visualized

If we're iterating over the data already
Other scenarios (see upcoming assignment 5)

List performance

<table>
<thead>
<tr>
<th>Operation</th>
<th>Arraylist</th>
<th>Singly linked list</th>
<th>Singly linked list (fast)</th>
<th>Doubly linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td>add to end</td>
<td>slow</td>
<td>slow</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>add to front</td>
<td>slow</td>
<td>slow</td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td>insert at index</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>remove</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>get</td>
<td>fast</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>set</td>
<td>fast</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>min/max</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>min/max from end</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>min/max from front</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td>size</td>
<td>fast</td>
<td>fast</td>
<td>fast</td>
<td>fast</td>
</tr>
</tbody>
</table>