Singly Linked Lists

Instructors: Sam McCauley and Dan Barowy

March 2, 2022

Admin

- Lab 3 in!
- Lab 4 out this afternoon
- Partners again! Fill out opt-out form *by this afternoon*
- New kind of lab: refactoring existing code
- Masks still required
- Remember to come on time Friday for the quiz
    - Look at graded responses; solutions on website under “Handouts”

Linked Lists

Motivating Example

- Let’s say we go to the movies

Motivating Example

- Let’s say we go to the movies
- But, the theater is very full. There are enough seats but we can’t all sit together

Motivating Example

- Let’s say we go to the movies
- But, the theater is very full. There are enough seats but we can’t all sit together
- How can I keep track of where all of you are sitting?

Motivating Example

- Let’s say we go to the movies
- But, the theater is very full. There are enough seats but we can’t all sit together
- How can I keep track of where all of you are sitting?
- One solution: keep a list of all of your seats

Motivating Example

- Let’s say we go to the movies
- But, the theater is very full. There are enough seats but we can’t all sit together
- How can I keep track of where all of you are sitting?
- One solution: keep a list of all of your seats
- Another option: I’ll keep track of one student. They keep track of the seat of the next student. So on and so forth

Motivating Example

- Let’s say we go to the movies
- But, the theater is very full. There are enough seats but we can’t all sit together
- How can I keep track of where all of you are sitting?
- One solution: keep a list of all of your seats
- Another option: I’ll keep track of one student. They keep track of the seat of the next student. So on and so forth
- Each student remembers the location of the next student (or none if they are the last). But I can still traverse all students!

Linked List

- A new kind of `List`.

Linked List

- A new kind of `List`.
    - So: will implement all the operations that a `Vector` implements

Linked List

- A new kind of `List`.
  - So: will implement all the operations that a `Vector` implements
  - (Looking ahead: will be faster for some operations; slower for others.)

Linked List

- A new kind of `List`.
    - So: will implement all the operations that a `Vector` implements
    - (Looking ahead: will be faster for some operations; slower for others.)
- Uses the principle from the theater example: each piece of data remembers the location of the next

First step: `Node`

- A *node* of a linked list stores one piece of data

First step: `Node`

- A *node* of a linked list stores one piece of data
- What does it need to store?

First step: `Node`

- A *node* of a linked list stores one piece of data
- What does it need to store?
- Needs to have the actual information

First step: `Node`

- A *node* of a linked list stores one piece of data
- What does it need to store?
- Needs to have the actual information
  - Probably of generic type

First step: `Node`

- A *node* of a linked list stores one piece of data
- What does it need to store?
- Needs to have the actual information
    - Probably of generic type
- Also needs to store the location of the next piece of data

First step: `Node`

- A *node* of a linked list stores one piece of data
- What does it need to store?
- Needs to have the actual information
  - Probably of generic type
- Also needs to store the location of the next piece of data
  - That is to say: needs to hold the next `Node`

Node

```
public class Node<E> {
   protected E data
   protected Node<E> nextElement;
    public Node(E v, Node<E> next) {
        data = v;
        nextElement = next;
    }
    public void setNext(Node<E> next) {
        nextElement = next;
    }
    public void setValue(E value) {
        data = value;
    }
```

```
    public Node<E> next() {
        return nextElement;
    }
    public E value() {
        return data;
    }
}
```

Creating a (Singly) Linked List

- We have a way to get from one `Node` to the next

Creating a (Singly) Linked List

- We have a way to get from one `Node` to the next
- What else do we need to store?

Creating a (Singly) Linked List

- We have a way to get from one `Node` to the next
- What else do we need to store?
    - First `Node`

Creating a (Singly) Linked List

- We have a way to get from one `Node` to the next
- What else do we need to store?
    - First `Node`
    - Maybe some other useful `List` information? Perhaps the number of stored items

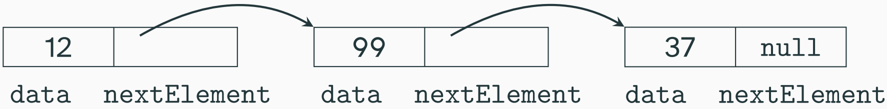
Creating a (Singly) Linked List

- We have a way to get from one `Node` to the next
- What else do we need to store?
    - First `Node`
    - Maybe some other useful `List` information? Perhaps the number of stored items
- And, then, need to implement methods

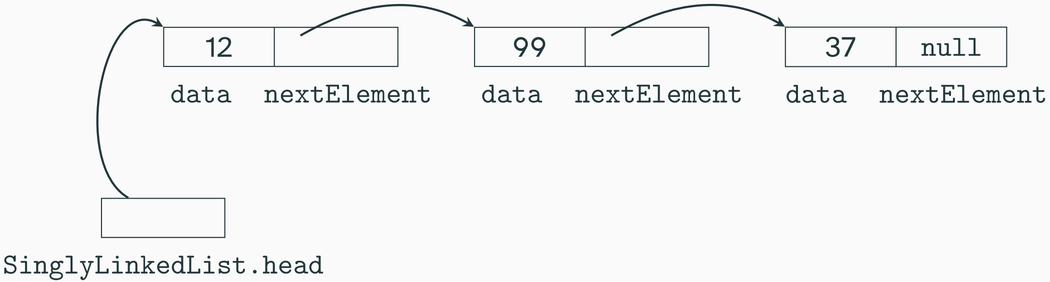
SinglyLinkedList

```
public class SinglyLinkedList<E>
{
    protected int count;                    // list size
    public Node<E> head; // ref. to first element

    public SinglyLinkedList()
    {
        head = null;
        count = 0;
    }

   //to fill in: methods...
```

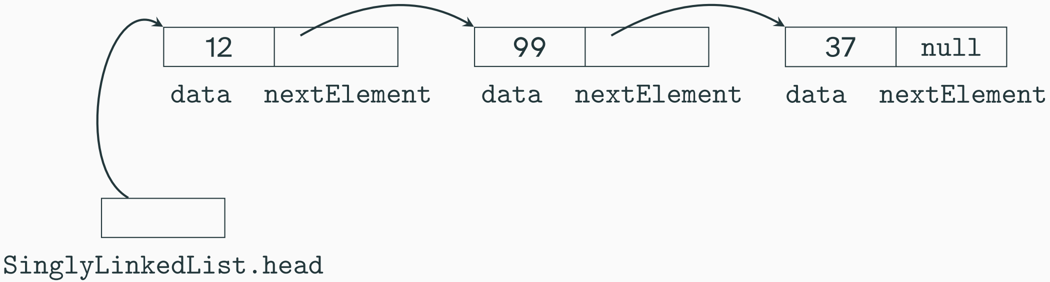
Where we are now



Where we are now

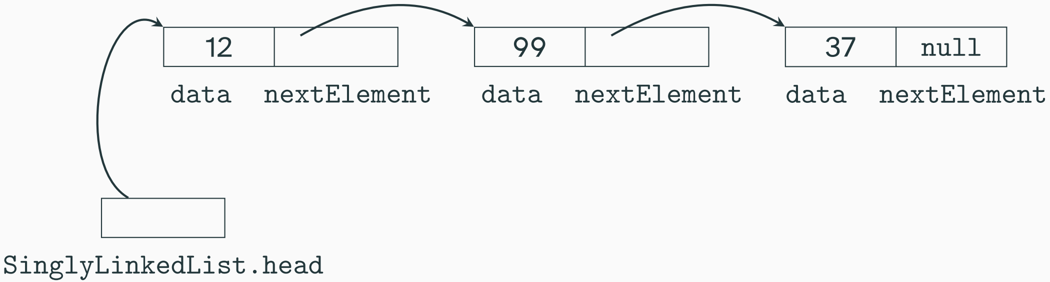


Where we are now



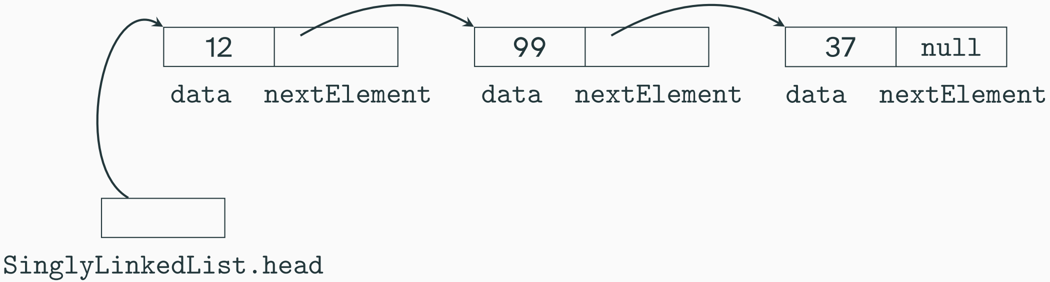
- Set of nodes linked to each other

Where we are now



- Set of nodes linked to each other
- We just need to store the address of the first one (`head`).

Where we are now

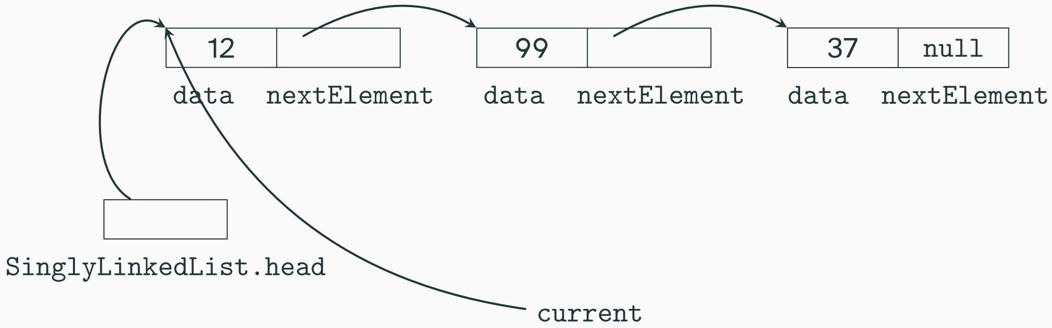


- Set of nodes linked to each other
- We just need to store the address of the first one (`head`).
- How can we perform `contains()`?

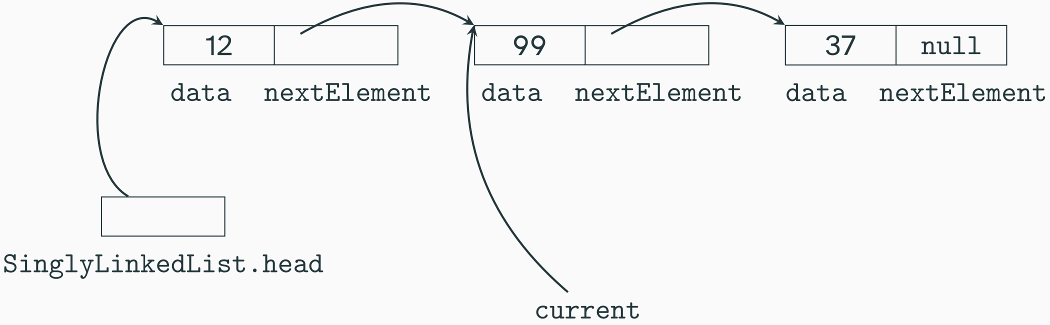
Draft of `contains`

```java
public boolean contains(E value) {
   Node<E> current = head;
   while(current != null) { //why == if these are objects?
      if(value.equals(current.value())) {
         return true;
      }
   }
   return false;
}
```

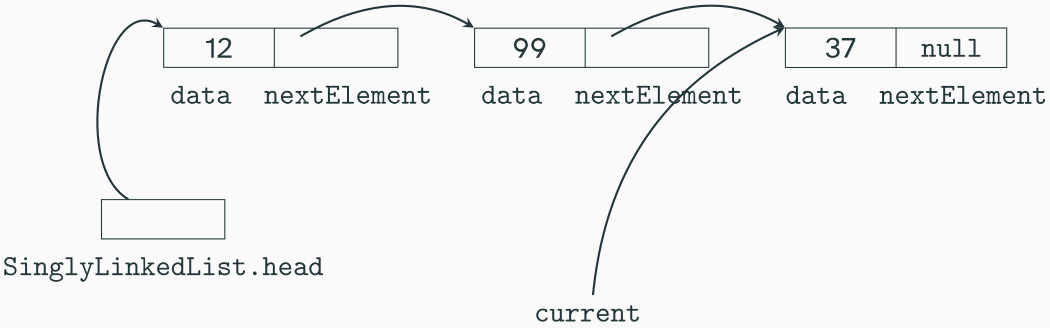
Contains Diagram



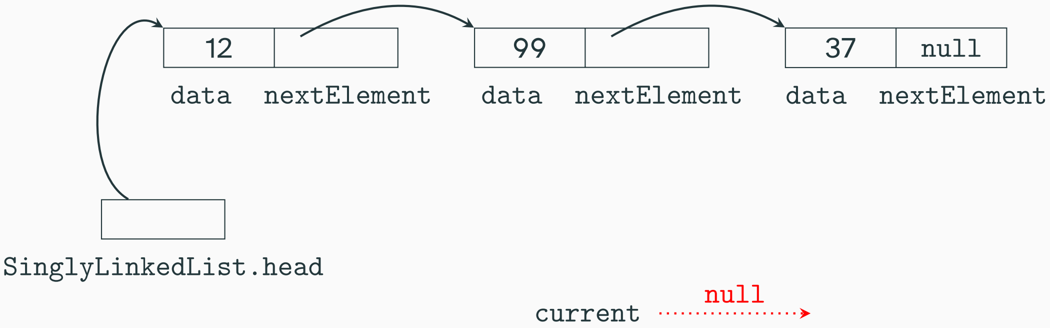
Contains Diagram



Contains Diagram



Contains Diagram



Plan for `add`?

- Where is the easiest place to add an item to our list?

Plan for `add`?

- Where is the easiest place to add an item to our list?
- To the front

Plan for `add`?

- Where is the easiest place to add an item to our list?
- To the front
- What do we need to do to add an item `value` to our list?

Plan for `add`?

- Where is the easiest place to add an item to our list?
- To the front
- What do we need to do to add an item `value` to our list?
    - Create a `Node<E>` that holds `value`. Let's call it `newNode`

Plan for `add`?

- Where is the easiest place to add an item to our list?
- To the front
- What do we need to do to add an item `value` to our list?
    - Create a `Node<E>` that holds `value`. Let's call it `newNode`
    - Set the `nextElement` of `newNode` to be the previous `head`

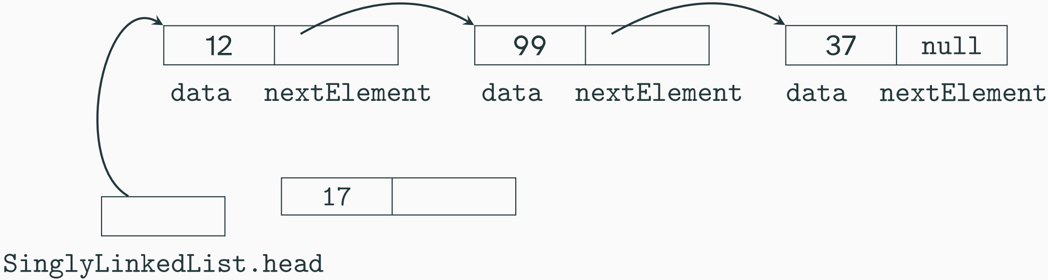
Plan for `add`?

- Where is the easiest place to add an item to our list?
- To the front
- What do we need to do to add an item `value` to our list?
  - Create a `Node<E>` that holds `value`. Let's call it `newNode`
  - Set the `nextElement` of `newNode` to be the previous `head`
  - Set the `head` to now point to `nextElement`

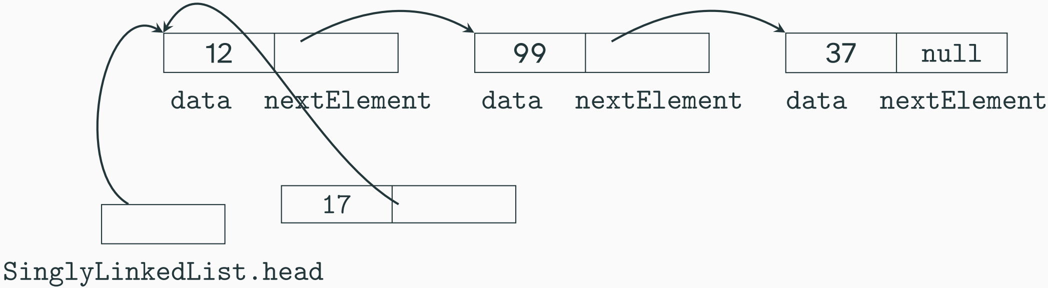
Plan for `add`?

- Where is the easiest place to add an item to our list?
- To the front
- What do we need to do to add an item `value` to our list?
    - Create a `Node<E>` that holds `value`. Let’s call it `newNode`
    - Set the `nextElement` of `newNode` to be the previous `head`
    - Set the `head` to now point to `nextElement`
- Also: update `count`

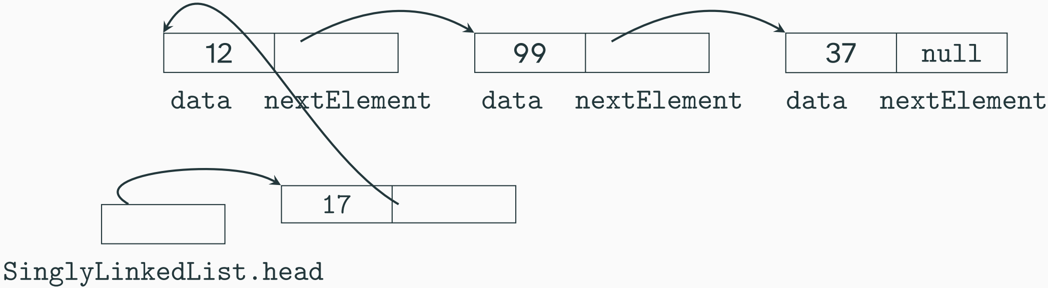
Add Diagram (Adding 17)



Add Diagram (Adding 17)



Add Diagram (Adding 17)



Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?

Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?
  - Only `head` and `count`!

Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?
  - Only `head` and `count`!
- What's the worst-case time for `add()`? (In terms of n, the length of the linked list.)

Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?
    - Only `head` and `count`!
- What's the worst-case time for `add()`? (In terms of n, the length of the linked list.)
    - $O(1)$. Just a constant number of operations. The length of the list never made a difference.

Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?
  - Only `head` and `count`!
- What's the worst-case time for `add()`? (In terms of n, the length of the linked list.)
  - $O(1)$. Just a constant number of operations. The length of the list never made a difference.
- How can we implement `set(int, E)` and `get(int)`?

Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?
  - Only `head` and `count`!
- What's the worst-case time for `add()`? (In terms of n, the length of the linked list.)
  - $O(1)$. Just a constant number of operations. The length of the list never made a difference.
- How can we implement `set(int, E)` and `get(int)`?
- How long do `set(int, E)` and `get(int)` take?

Singly Linked List Discussion

- What does the `SinglyLinkedList` object itself consist of?
    - Only `head` and `count`!
- What's the worst-case time for `add()`? (In terms of n, the length of the linked list.)
    - $O(1)$. Just a constant number of operations. The length of the list never made a difference.
- How can we implement `set(int, E)` and `get(int)`?
- How long do `set(int, E)` and `get(int)` take?
- If I have a `Node<E>` in the middle of my list, how long does it take to add a new node *after* it?

Motivating Recursion (Induction Intro)

Goal of this section

- Chat about what makes a recursive algorithm correct

Goal of this section

- Chat about what makes a recursive algorithm correct

- We’ll get more formal about this on Friday

Finding number of X in a string

```
public static int numX(String s) {
   if(s.length() == 0) {
      return 0;
   }
   if(s.charAt(s.lengt()-1) == 'X') {
      return 1 + numX(s.substring(0,s.length() - 1));
   else {
      return numX(s.substring(0,s.length() - 1));
   }
}
```

Finding number of X in a string

```java
public static int numX(String s) {
   if(s.length() == 0) {
      return 0;
   }
   if(s.charAt(s.lengt()-1) == 'X') {
      return 1 + numX(s.substring(0,s.length() - 1));
   else {
      return numX(s.substring(0,s.length() - 1));
   }
}
```

How do we know that this method works correctly?

Where to start?

```
public static int numX(String s) {
   if(s.length() == 0) {
      return 0;
   }
   if(s.charAt(s.lengt()-1) == 'X') {
      return 1 + numX(s.substring(0,s.length() - 1));
   else {
      return numX(s.substring(0,s.length() - 1));
   }
}
```

Where to start?

```
public static int numX(String s) {
   if(s.length() == 0) {
      return 0;
   }
   if(s.charAt(s.lengt()-1) == 'X') {
      return 1 + numX(s.substring(0,s.length() - 1));
   else {
      return numX(s.substring(0,s.length() - 1));
   }
}
```

- If `s` has length 0, then this algorithm correctly returns 0.

Where to start?

```
public static int numX(String s) {
   if(s.length() == 0) {
      return 0;
   }
   if(s.charAt(s.lengt()-1) == 'X') {
      return 1 + numX(s.substring(0,s.length() - 1));
   else {
      return numX(s.substring(0,s.length() - 1));
   }
}
```

- If `s` has length 0, then this algorithm correctly returns 0.
- What if `s` has length 1?

Proving recursive correctness

- If `s` has length 1:

Proving recursive correctness

- If `s` has length 1:
    - We know that `numX(s.substring(0,0))` returns `0` because we know that `numX` is correct on strings of length `0`.

Proving recursive correctness

- If `s` has length 1:
  - We know that `numX(s.substring(0,0))` returns 0 because we know that `numX` is correct on strings of length 0.
  - If the first character is `X`, we return $1 + 0 = 1$.

Proving recursive correctness

- If `s` has length 1:
  - We know that `numX(s.substring(0,0))` returns 0 because we know that `numX` is correct on strings of length 0.
  - If the first character is `X`, we return $1 + 0 = 1$.
  - If the first character is not `X`, we return $0 + 0 = 0$.

Proving recursive correctness

- If `s` has length 1:
    - We know that `numX(s.substring(0,0))` returns 0 because we know that `numX` is correct on strings of length 0.
    - If the first character is `X`, we return $1 + 0 = 1$.
    - If the first character is not `X`, we return $0 + 0 = 0$.
    - In both cases we're correct.

Proving recursive correctness

- If `s` has length 1:
  - We know that `numX(s.substring(0,0))` returns 0 because we know that `numX` is correct on strings of length 0.
  - If the first character is `X`, we return $1 + 0 = 1$.
  - If the first character is not `X`, we return $0 + 0 = 0$.
  - In both cases we're correct.
- So `numX()` works on strings of length at most 1.

Proving recursive correctness

- If `s` has length 2:

Proving recursive correctness

- If `s` has length 2:
  - We know that `numX(s.substring(0,1)` is correct because we know that `numX` is correct on strings of length 1.

Proving recursive correctness

- If `s` has length 2:
  - We know that `numX(s.substring(0,1)` is correct because we know that `numX` is correct on strings of length 1.
  - We add 1 if the first character is `'X'`, 0 otherwise. So we are correct for strings of length at most 2

Proving recursive correctness

- If `s` has length 2:
  - We know that `numX(s.substring(0,1)` is correct because we know that `numX` is correct on strings of length 1.
  - We add 1 if the first character is `'X'`, `0` otherwise. So we are correct for strings of length at most 2
- So `numX()` works on strings of length at most 2.

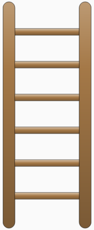
Our strategy

- Start with base case

Our strategy

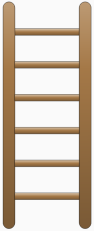
- Start with base case
- Slowly argue that it works for larger and larger strings

Visualizing our Strategy



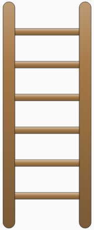
- How can we climb to the top of a ladder?

Visualizing our Strategy



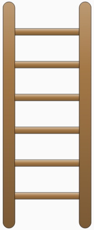
- How can we climb to the top of a ladder?
- Here’s a two step process:

Visualizing our Strategy



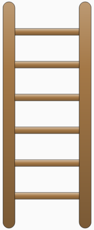
- How can we climb to the top of a ladder?
- Here’s a two step process:
  - Figure out how to get on some rung of the ladder

Visualizing our Strategy



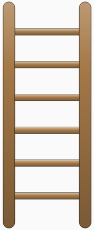
- How can we climb to the top of a ladder?
- Here's a two step process:
    - Figure out how to get on some rung of the ladder
    - Figure out a method to get from one rung to the next rung

Visualizing our Strategy



- How can we climb to the top of a ladder?
- Here's a two step process:
    - Figure out how to get on some rung of the ladder
    - Figure out a method to get from one rung to the next rung
- If I do both of these, will I always make it to the top of the ladder?

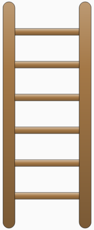
Our strategy



Ladder analogy: each step of the ladder is the length of the string in our recursive method. We want to show that our method is correct for a string of a certain length.

- Start with base case

Our strategy

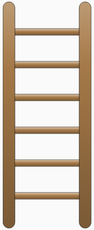


Ladder analogy: each step of the ladder is the length of the string in our recursive method. We want to show that our method is correct for a string of a certain length.

- Start with base case

Figure out how to get on the ladder

Our strategy

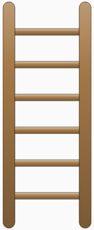


Ladder analogy: each step of the ladder is the length of the string in our recursive method. We want to show that our method is correct for a string of a certain length.

- Start with base case *Figure out how to get on the ladder*

- Slowly argue that it works for larger and larger strings

Our strategy



Ladder analogy: each step of the ladder is the length of the string in our recursive method. We want to show that our method is correct for a string of a certain length.

- Start with base case *Figure out how to get on the ladder*
- Slowly argue that it works for larger and larger strings *From one rung of the ladder, how can we get to the next rung?*