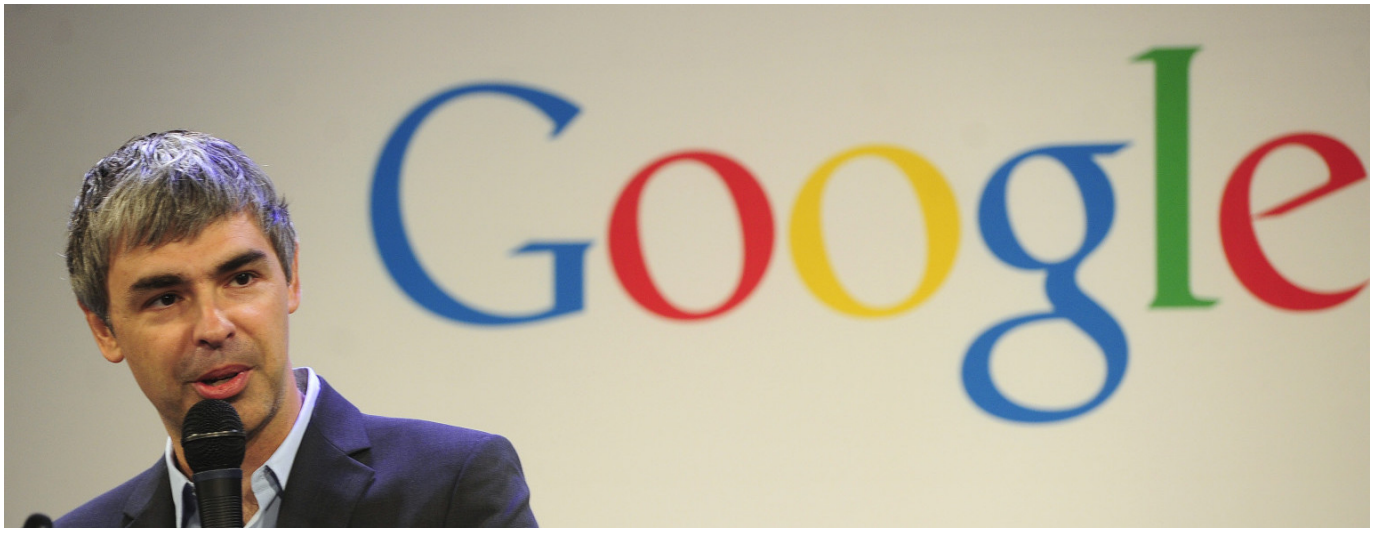
22 Address Bloc: Searching



> “The ultimate search engine would basically understand everything in the world, and it would always give you the right thing. And we're a long, long ways from that.”
>
> — Larry Page, cofounder of Google Inc.

Introduction

What use is an address book if it's not searchable? Let's add the ability to search Address Bloc.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint for details**.

We'll use a technique called *binary search* to implement our search functionality. We'll learn more about binary search as this checkpoint continues.

Test

Let's create some tests that will help define `binary_search`'s behavior. Since we are testing a method that is a part of `AddressBook`, our tests belong in `address_book_spec.rb`:

spec/address_book_spec.rb

```
  ...

+   # Test the binary_search method
+   describe "#binary_search" do
+     it "searches AddressBook for a non-existent entry" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Dan")
+       expect(entry).to be_nil
+     end
+   end

  ...
```

Run the test and verify that it fails:

Terminal

```
$ rspec spec/address_book_spec.rb
..........F

Failures:

  1) AddressBook#binary_search searches AddressBook for a non-existent entry
     Failure/Error: entry = book.binary_search("Dan")

     NoMethodError:
       undefined method `binary_search' for #<AddressBook:0xc756ab3>
     # ./spec/address_book_spec.rb:94:in `(root)'
```

We see an undefined method error since `binary_search` is undefined.

Stub

Let's add the **stub** of `binary_search` to `AddressBook`. It will return `nil` for now:

models/address_book.rb

```
  ...

+   # Search AddressBook for a specific entry by name
+   def binary_search(name)
+   end

  ...
```

Test Again

Run the test again. The test searches for `Entry` "Dan" that does not exist and our stubbed out `binary_search` returned `nil`, thus the test passes:

Terminal

```
$ rspec spec/address_book_spec.rb
............

Finished in 0.0037 seconds (files took 0.10661 seconds to load)
12 examples, 0 failures
```

More Tests

Let's add another test:

```
  ...

    # Test the binary_search method
    describe "#binary_search" do
      it "searches AddressBook for a non-existent entry" do
        book.import_from_csv("entries.csv")
        entry = book.binary_search("Dan")
        expect(entry).to be_nil
      end

+     it "searches AddressBook for Bill" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Bill")
+       expect(entry).to be_a Entry
+       check_entry(entry, "Bill", "555-555-4854", "bill@blocmail.com")
+     end
    end

  ...
```

We added a test for "Bill". We expect `binary_search` to return an object of type `Entry`. We also use `check_entry` to validate our expectation that this object has its attributes set properly. Run the tests again:

Terminal

```
$ rspec spec/address_book_spec.rb
............F

Failures:

  1) AddressBook#binary_search searches AddressBook for Bill
     Failure/Error: expect(entry).to be_a Entry
       expected nil to be a kind of Entry
     # ./spec/address_book_spec.rb:101:in `(root)'

Finished in 0.204 seconds (files took 0.608 seconds to load)
13 examples, 1 failure

Failed examples:

rspec ./spec/address_book_spec.rb:98 # AddressBook#binary_search searches AddressBoo
```

Our new test fails since `binary_search` still returns `nil`, but it will pass when we write the implementation of the method.

Repeat this pattern for the other entries in `entries.csv`:

```
  ...

+     it "searches AddressBook for Bob" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Bob")
+       expect(entry).to be_a Entry
+       check_entry(entry, "Bob", "555-555-5415", "bob@blocmail.com")
+     end
+
+     it "searches AddressBook for Joe" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Joe")
+       expect(entry).to be_a Entry
+       check_entry(entry, "Joe", "555-555-3660", "joe@blocmail.com")
+     end
+
+     it "searches AddressBook for Sally" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Sally")
+       expect(entry).to be_a Entry
+       check_entry(entry, "Sally", "555-555-4646", "sally@blocmail.com")
+     end
+
+     it "searches AddressBook for Sussie" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Sussie")
+       expect(entry).to be_a Entry
+       check_entry(entry, "Sussie", "555-555-2036", "sussie@blocmail.com")
+     end

  ...
```

Finally, add a test for an entry that is similar to something that we know exists, but not exactly the same:

```
  ...

+     it "searches AddressBook for Billy" do
+       book.import_from_csv("entries.csv")
+       entry = book.binary_search("Billy")
+       expect(entry).to be_nil
+     end

  ...
```

Let's run our spec again and see the list of failing tests:

Terminal

```
$ rspec spec/address_book_spec.rb
............FFFFF

Failures:

  1) AddressBook#binary_search searches AddressBook for Bill
     Failure/Error: expect(entry).to be_a Entry
       expected nil to be a kind of Entry
     # ./spec/address_book_spec.rb:101:in `(root)'

  2) AddressBook#binary_search searches AddressBook for Bob
     Failure/Error: expect(entry).to be_a Entry
       expected nil to be a kind of Entry
     # ./spec/address_book_spec.rb:108:in `(root)'

  3) AddressBook#binary_search searches AddressBook for Joe
     Failure/Error: expect(entry).to be_a Entry
       expected nil to be a kind of Entry
     # ./spec/address_book_spec.rb:115:in `(root)'

  4) AddressBook#binary_search searches AddressBook for Sally
     Failure/Error: expect(entry).to be_a Entry
       expected nil to be a kind of Entry
     # ./spec/address_book_spec.rb:122:in `(root)'

  5) AddressBook#binary_search searches AddressBook for Sussie
     Failure/Error: expect(entry).to be_a Entry
       expected nil to be a kind of Entry
     # ./spec/address_book_spec.rb:129:in `(root)'

Finished in 0.257 seconds (files took 0.649 seconds to load)
17 examples, 5 failures
...
```

Implement

Since the `add_entry` method inserts items alphabetically, we can use a search algorithm that is optimal for sorted lists. Many **search algorithms** exist, but we'll implement **binary search** in Address Bloc. `binary_search` uses a **divide and conquer** design pattern. Add the following code to implement `binary_search`:

address_book.rb

```
    # Search AddressBook.entries for a specific entry by name
    def binary_search(name)
  # #1
+     lower = 0
+     upper = entries.length - 1

  # #2
+     while lower <= upper
  # #3
+       mid = (lower + upper) / 2
+       mid_name = entries[mid].name
+
  # #4
+       if name == mid_name
+         return entries[mid]
+       elsif name < mid_name
+         upper = mid - 1
+       elsif name > mid_name
+         lower = mid + 1
+       end
+     end

  # #5
      return nil
    end
```

At **#1**, we save the index of the leftmost item in the array in a variable named `lower`, and the index of rightmost item in the array in `upper`. If we think of the array in terms of left-to-right where the leftmost item is the zeroth index and the rightmost item is the `entries.length-1` index.

At **#2**, we loop while our `lower` index is less than or equal to our `upper` index.

At **#3**, we find the middle index by taking the sum of `lower` and `upper` and dividing it by two. Ruby will truncate any decimal numbers, so if `upper` is five and `lower` is zero then `mid` will get set to two. Then we retrieve the name of the entry at the middle index and store it in `mid_name`.

At **#4**, we compare the name that we are searching for, `name`, to the name of the middle index, `mid_name`. We use the `==` operator when comparing the names which makes the

search **case sensitive**

- If `name` is equal to `mid_name` we've found the name we are looking for so we return the entry at index `mid`.
- If `name` is alphabetically before `mid_name`, then we set `upper` to `mid - 1` because the name must be in the lower half of the array.
- If `name` is alphabetically after `mid_name`, then we set `lower` to `mid + 1` because the name must be in the upper half of the array.

At **#5**, if we divide and conquer to the point where no match is found, we return `nil`.

Running the specs a final time shows them all passing:

Terminal

```
rspec spec/address_book_spec.rb
.................

Finished in 0.326 seconds (files took 1.23 seconds to load)
17 examples, 0 failures
```

The following video elaborates on binary search in greater detail:

Let's walk through `binary_search` using the values in `entries.csv` as our sorted list and search for "Bill".

Step One: binary_search is called with name set to "Bill"

"Bill" is at the zeroth index of **entries.**
"Bob" is at the first index of **entries.**
"Joe" is at the second index of **entries.**
"Sally" is at the third index of **entries.**
"Sussie" is at the fourth index of **entries.**

name == Bill **entries** = ["Bill", "Bob", "Joe", "Sally", "Sussie"]

lower == 0 **upper** == 4, since (**entries.length - 1**) == 4 on **line 3**

```
1. def binary_search(name)
2.   lower = 0
3.   upper = entries.length - 1
4.     while lower <= upper
5.       mid = (lower + upper) / 2
6.       mid_name = entries[mid].name
7.
8.       if name == mid_name
9.         return entries[mid]
10.       elsif name < mid_name
11.        upper = mid - 1
12.       elsif name > mid_name
13.         lower = mid + 1
14.       end
15.     end
16.
17.    return nil
18.  end # End binary_search()
```

Step Two: we step through the while loop for the first time (**iteration one**)

name == Bill **entries** = ["Bill", "Bob", "Joe", "Sally", "Sussie"]

lower == 0 **upper** == 4

mid == 2 — **mid** gets set to two since (**upper** + **lower**) / 2 == 2 on **line 6**

mid_name == "Joe" — **mid_name** gets set to "Joe" since **entries[2]** == "Joe" on **line 7**

upper == 1 — **upper** now gets set to **mid - 1** since **name,** "Bill", is lexicographically earlier than **mid_name,** "Joe", on **lines 12 and 13**

```
1. def binary_search(name)
2.   lower = 0
3.   upper = entries.length - 1
4.     while lower <= upper
5.       mid = (lower + upper) / 2
6.       mid_name = entries[mid].name
7.
8.       if name == mid_name
9.         return entries[mid]
10.       elsif name < mid_name
11.        upper = mid - 1
12.       elsif name > mid_name
13.         lower = mid + 1
14.       end
15.     end
16.
17.    return nil
18.  end # End binary_search()
```

Step Three: we step through the while loop for the second time (**iteration two**)

name == Bill **entries** = ["Bill", "Bob", "Joe", "Sally", "Sussie"]

lower == 0 **upper** == 1

mid == 0 — **mid** gets set to zero since (**upper** + **lower**) / 2 == 0 on **line 6.**

1 divided by 2 is equals zero since the remainder is 1 and it's thrown out by ruby.

mid_name == "Bill" — **mid_name** gets set to "Bill" since **entries[0]** == "Bill" on **line 7** and then binary_search returns "Bill" since it found a match on **line 8**

```
1. def binary_search(name)
2.   lower = 0
3.   upper = entries.length - 1
4.     while lower <= upper
5.       mid = (lower + upper) / 2
6.       mid_name = entries[mid].name
7.
8.       if name == mid_name
9.         return entries[mid]
10.       elsif name < mid_name
11.        upper = mid - 1
12.       elsif name > mid_name
13.         lower = mid + 1
14.       end
15.     end
16.
17.    return nil
18.  end # End binary_search()
```

`binary_search` took three steps to find "Bill" with five entries.

Recap

Concept	Description

TDD	We stubbed out `binary_search` to act as a placeholder. Then we built tests to define the expected behavior of `binary_search`.
Binary Search	With our tests in place, we built the implementation of `binary_search` until the tests passed.

22. Address Bloc: Searching

Assignment	Discussion	Submission

Create a new Git feature branch for this assignment. See **Git Checkpoint Workflow: Before Each Assignment** for details.

As we alluded to, many search algorithms exist. Your assignment is to create a method that performs an iterative search:

- Start by stubbing `iterative_search` in `address_book.rb`.
- Create tests for `iterative_search` in `address_book_spec.rb`. You can use the same pattern we used for testing `binary_search`.
- Fill in the body of `iterative_search`:

> Starting from the first entry in `AddressBook.entries`, iterate over the entries until you find the match. When the match is found, return it. If no match is ever found, return `nil`.

- Ensure the new tests pass with your `iterative_search` method.

Commit your assignment in Git. See **Git Checkpoint Workflow: After Each Assignment** for details. Submit your commit to your mentor.

After submitting your commit, consider the following questions:

- Which algorithm is more efficient: binary search or iteration? How do you know?
- How might you measure the difference?

Message your mentor with your answers (at the very least, your thoughts) to these questions.

Solution

Do not watch this video until after you've attempted to complete the assignment. If you struggle to complete the assignment, submit your best effort to your mentor *before watching a solution video*.

Media Queries Solution

assignment completed

?