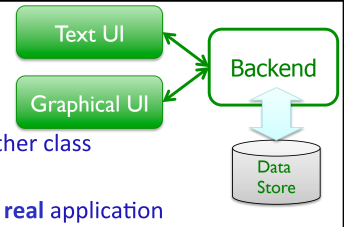
Objectives

- Search strategies
  - Review
  - Extensions
- Broader Issue: Text Analysis

Reviewing Lab 10



- Created two classes
  - Used one class within another class
  - Tested them
  - Example of a backend to a **real** application
    - Could add a different user interface
- “Good judgment comes from experience”
  - Test methods after writing it
  - Remember your data types
  - Refer to the data type’s API
- What could you do to improve your development process?

Review

- We discussed two different search techniques:
  - What were they?
  - How do they compare?

Review: Search Using `in` Review

- Iterates through a list, checking if the element is found
- Known as ***linear search***
- **Implementation:**

```
def linearSearch(searchlist, key):
   for elem in searchlist:
      if elem == key:
         return True
   return False
```

value	8	5	3	7
pos	0	1	2	3

What are the strengths and weaknesses of implementing search this way?

search.py

Review: Linear Search

- **Overview**: Iterates through a list, checking if the element is found
- **Benefits:**
  - Works on *any* list
- **Drawbacks**:
  - **Slow**, on average: needs to check each element of list if the element is not in the list

Review: Binary Search: Eliminate Half the Possibilities

- Repeat until find value (or looked through all values)
  - Guess middle *value* of possibilities
    - (not middle *position*)
  - If match, found!
  - Otherwise, find out too high or too low
  - Modify your possibilities
    - Eliminate the possibilities from your number and higher/lower, as appropriate
- Known as **Binary Search**

Binary Search Implementation

```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         low = mid+1
      else:
         high = mid-1
```

Our condition?
What if not found?

search2.py

Binary Search Implementation

```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while low <= high :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid    # return True
      elif key > searchlist[mid]:
         low = mid+1
      else:
         high = mid-1
   return -1    # return False
```

If you just want to know if it's in the list

search2.py

Binary Search

- Example of a ***Divide and Conquer*** algorithm
  - Break into smaller pieces that you can solve
- Benefits:
  - Faster to find elements (especially with larger lists)
- Drawbacks:
  - Requires that data can be compared
    - `__lt__, __eq__` methods implemented by the class (or another solution)
  - List **must** be sorted before searching
    - Takes time to sort

Key Questions in Computer Science

- How can we efficiently organize data?
- How can we efficiently search for data, given various constraints?
  - Example: data may or may not be sortable
- What are the tradeoffs?

Empirical Study of Search Techniques

Goal: Determine which technique is better under various circumstances

- How long does it take to find various keys?
  - **Measure** by the number of comparisons
  - Vary the size of the list and the keys
  - What are good tests for the lists and the keys?

`search_compare.py`

Empirical Study of Search Techniques

- Analyzing Results …
  - By how much did the number of comparisons for *linear search* vary?
  - By how much did the number of comparisons for *binary search* vary?
- What conclusions can you draw from these results?

`search_compare.py`

Search Strategies Summary

- Which search strategy should I use under the following circumstances?
  - I have a short list
  - I have a long list
  - I have a long sorted list

Search Strategies Summary

- Which search strategy should I use under the following circumstances?
  - I have a short list
    - How short? How many searches? Linear (`in`)
  - I have a long list
    - Linear (`in`) - because don't know if in order, comparable
    - Alternatively, may want to sort the list and *then* perform binary search, if sorting first won't be more effort than just sorting.
  - I have a long sorted list
    - Binary

Extensions to Search

In FaceSpace, we want to find people who have a certain name.

Consider what happens when `searchlist` is a list of *Persons* and key is a name (a `str`)

We want to find a `Person` whose name matches the key and return the *Person*

List of `Person` objects

0	1	2	3	4
Person Id:"1" "Henry"	Person Id:"2" "Natalie"	Person Id:"3" "Chris"	Person Id: "4" "Ben"	Person Id: "5" "Samuel"

Example: looking for a person with the name "Chris"…

List of Person objects

0	1	2	3	4
Person Id:"1" "Henry"	Person Id:"2" "Natalie"	Person Id:"3" "Chris"	Person Id: "4" "Ben"	Person Id: "5" "Samuel"

0	1	2	3	4
Person Id: "4" "Ben"	Person Id:"3" "Chris"	Person Id: "1" "Henry"	Person Id:"2" "Natalie"	Person Id:"5" "Samuel"

Sorted by name using:

```
personList.sort(key=Person.getName)
```

Extensions to Solution

Consider what happens when **searchlist** is a list of *Persons*, ***key*** is a ***str*** representing the name
- Goal: find a person with a certain name

```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while low <= high :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         low = mid+1
      else:
         # look in lower half
         high = mid-1
   return -1
```

0	1	2	3	4
Person Id: "4" "Ben"	Person Id:"3" "Chris"	Person Id: "1" "Henry"	Person Id:"2" "Natalie"	Person Id:"5" "Samuel"

Extensions to Solution

Consider what happens when **searchlist** is a list of *Persons*, ***key*** is a ***str*** representing the name
- Goal: find a *person* with a certain network

What can we do to make search results more intuitive?

```
def search(searchlist, key):
   low=0
   high = len(searchlist)-1
   while low <= high :
      mid = (low+high)//2
      if searchlist[mid] == key:
         return mid
      elif key > searchlist[mid]:
         # look in upper half
         low = mid+1
      else:
         # look in lower half
         high = mid-1
   return -1
```

0	1	2	3	4
Person Id: "4" "Ben"	Person Id:"3" "Chris"	Person Id: "1" "Henry"	Person Id:"2" "Natalie"	Person Id:"5" "Samuel"

Summary of Extensions to Solution

- Check the *name* of the Person at the midpoint
- Represent, handle when no Person matches
- What could we do if more than one person has that name?

- Note: we're not implementing "name contains"
  - How could we implement that?

Broader Issue

Burke	Alex	John	Anna Kate	Buddy
Charlotte	Austin	Josette	Lexi	Mike
Collin	George	Leslie	Robert	Mira
Victor	Molly	Win	Sarah	Tony
Zander				

Digital Humanities: Text Analysis

- What were the most interesting/surprising questions asked/answered?
- What are new questions you would like answered?
  - Could you implement those with what you currently know?

Google n-grams

- https://books.google.com/ngrams

Looking Ahead

- Lab 11
  - Extensions to FaceSpace
- Broader Issue: Social Media algorithms