Objectives

- Lab 10 Review
- Search strategies

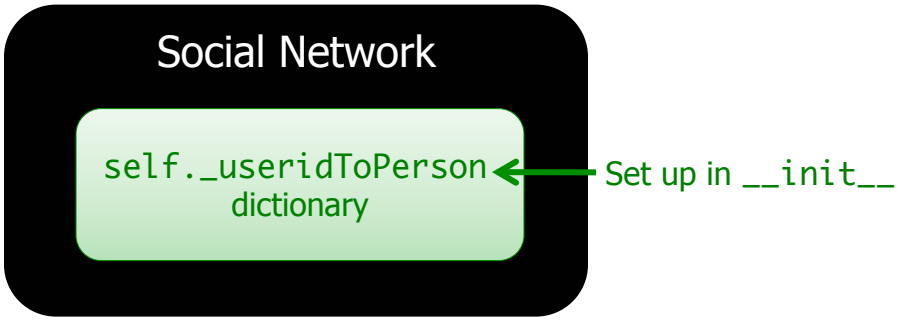
Lab 10

- Trying to solve a real problem
- Started with designing the solution from a vague specification
- Broke into smaller problems (different classes, different responsibilities)
- Implementing smaller components
  - Following the specification
- Building to large component

Lab 10 Discussion

- How can we call other methods of the data type when we're in one method of the data type?
  - Example: If I'm in the `__str__(self)` method of the Person class, how can I call the `getNumFriends()` method?
- How do the `SocialNetwork` class and `Person` class work together?

SocialNetwork



Do I need to do operations on the dictionary?
- Then operate on `self._useridToPerson`

Do I need to do operations on a `SocialNetwork`?
- Then, call methods on `self`.

The Common Conundrum

- You have a large tool box.
- You need to keep track of all the tools you have in your box
  - You will be combining a variety of tools in different ways

This is **Problem Solving**!

The Common Conundrum

- You have a large tool box.
- You need to keep track of all the tools you have in your box
  - You will be combining a variety of tools in different ways

This is **Problem Solving**!

- How can you figure out what tool to use?
  - How am I representing this information? What is its type?
  - What operations/methods/functions are available?
  - When I ran into this situation before, how did I solve it?
  - How can I make it clearer what is going on?

Lab 10 FAQ for common issues

References

- Check out the slides for lab10
  - Hints on reading in files
- Lab 10 FAQ
- What problem is this similar to?

- Student assistant 7-9 p.m. Thurs

SEARCHING

Search Using `in`

- 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

pos

8	5	3	7
0	1	2	3

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

Linear Search

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

High-Low Game/TPIR Clock Game

- I'm thinking of a number between 1-100
- You want to guess the number as quickly as possible, i.e., in fewest guesses
- For every number you guess, I'll tell you if you got it right. If you didn't, I'll tell you whether you're too high or too low

Reminder: write down guesses

High-Low Game/TPIR Clock Game

- I'm thinking of a number between 1-100
- You want to guess the number as quickly as possible, i.e., in fewest guesses
- For every number you guess, I'll tell you if you got it right. If you didn't, I'll tell you whether you're too high or too low

➔What is your best guessing strategy?

Strategy: Eliminate Half the Possibilities

- Repeat until find value or looked through all values
  - Guess middle value of possibilities
  - 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***

Searching…

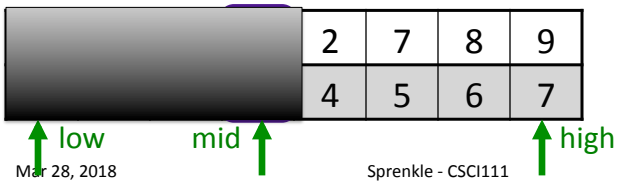
value	-3	0	0	1	2	7	8	9
pos	0	1	2	3	4	5	6	7

Use algorithm to search for key = 8

Searching for 8

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

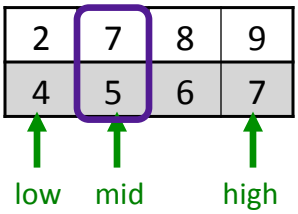
- Find the middle of the list
  - Positions: 0-7, so mid position is ((7+0)//2) = 3
- Check if the key equals the value at mid (1)
  - If so, report the location
- Check if the key is higher or lower than value at mid
  - Search the appropriate half of the list



8 > 1, so look in upper half

Searching for 8

- mid is 5 ((7+4)//2), list[5] is 7



8>7,
so look in upper half

Searching for 8

- mid is 5 ((7+4)//2), list[5] is 7

2	7	8	9
4	5	6	7

8>7,
so look in upper half

- mid is 6 ((7+6)//2), list[6] is 8

8	9
6	7

8==8,
FOUND IT at position 6!

What if searched for 6 instead of 8?

Searching for 6

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

- Will follow same execution flow, but 6 is not in the list
- mid is 6, list[5] is 7

2	7	8	9
4	5	6	7

6 < 7, so will try to look in lower half of the list

- mid is 4, list[4] is 2

2
4

6>2, so will try to look in
upper half of the list,
but we've already determined it's not there.
How do we know to stop looking?

Implementation Group Work

```
def search(searchlist, key):
    """Pre: searchlist is a list of
integers in sorted order.  Returns the
position of key (an integer) in the list
of integers (searchlist) or -1 if not
found"""
```

- Trace through your program using examples
  - Start simple (small lists)
  - Do what the program says *exactly*, not what you *think* the program says

One Solution

```
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

One Solution

Cutting list in half
Discuss tradeoffs

```
def altBinarySearch(searchlist, key):
    # Base Case: ran out of elements in the list
    if len(searchlist) == 0:
        return NOT_FOUND

    low = 0
    high = len(searchlist)-1
    mid = (low+high)//2

    valueAtMid = searchlist[mid]
    if valueAtMid == key:
        return mid
    if low == high:
        return NOT_FOUND

    if searchlist[mid] < key: # search upper half
        return altBinarySearch(searchlist[mid+1:], key)
    else: # search lower half
        return altBinarySearch(searchlist[:mid], key)
```

Creating a new list
Additional memory use

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)
- Limitations:
  - Requires that data can be compared
    - `__lt__, __eq__` methods implemented by the class
  - List **must** be sorted before searching
    - Takes time to sort beforehand

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`

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?

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
  - I have a long sorted list
    - Binary

Schedule

- No Broader Issue for Friday
  - Push to Friday of next week – Facebook and data
- Lab 10 – due Friday