

## Objectives

- Search strategies
- Exceptions
- Broader Issue: One Laptop Per Child

Mar 28, 2007

Sprenkle - CS111

1

## Search Using **in** Review

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

```
def inSearch(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?

Mar 28, 2007

Sprenkle - CS111

search.py

2

## Linear Search

- **Overview:** Iterates through a list, checking if the element is found
- **Benefits:**
  - Works on *any* list
- **Drawbacks:**
  - Does not tell us where in the list it is
    - What if wanted to do something to that element?
    - Could implement our own version that returns the position
  - Slow -- needs to check **each** element of list if the element is not in the list

Mar 28, 2007

Sprenkle - CS111

3

## Strategy: 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**

Mar 28, 2007

Sprenkle - CS111

4

## A Binary Search 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

Mar 28, 2007

Sprenkle - CS111

search2.py

5

## 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
    - `__cmp__` method implemented by the class
  - List **must** be sorted before searching
    - Takes time to search

Mar 28, 2007

Sprenkle - CS111

6

## 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?

Mar 28, 2007

Sprenkle - CS111 `search_compare.py` 7

## Empirical Study of Search Techniques

- 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?

Mar 28, 2007

Sprenkle - CS111 `search_compare.py` 8

## Modifying Solution

```
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
```

What if we had a list of Cards instead of a list of integers?

- What needs to be changed?
- What has to be done/verified in the Card class?

Mar 28, 2007

Sprenkle - CS111

9

## Extensions to Solution

```
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]:
            low=mid+1
        else:
            high = mid-1
    return -1
```

Consider what happens when **searchlist** is a list of *Songs*

- What if we wanted to check if the song's *title* matched the key and return the *song*?

Mar 28, 2007

Sprenkle - CS111

10

## Extensions to Solution

```
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]:
            low=mid+1
        else:
            high = mid-1
    return -1
```

Consider what happens when **searchlist** is a list of *Songs*

- What if we wanted **all** the songs with the title that matched the key?

Mar 28, 2007

Sprenkle - CS111

11

## Summary of Extensions to Solution

- Check the **title** of the Song at the midpoint
- Get the songs before and after that song in the list that have the same title and put in a list
- Represent, handle when no song matches
- For “most intuitive” results:
  - Strip, lowercase the key
    - Which means what for your algorithm?
- Note: we’re not just implementing “title contains”
  - How could we implement that?

Mar 28, 2007

Sprenkle - CS111

12

## Search Strategies Summary

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

Mar 28, 2007

Sprenkle - CS111

13

## Search Strategies Summary

- Which search strategy should I use under the various 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

Mar 28, 2007

Sprenkle - CS111

14

## Validating User Input

```
def main():
    #Program mission statement
    print "This program determines your birth year"
    print "given your age and the current year \n"

    age=input("Enter your age: ")
    currentyear=input("Enter the current year: ")

    #Subtract age from current year
    birthyear=currentyear - age
    #Display output to the user
    print "You were either born in", birthyear, "or", birthyear-1
```

Mar 28, 2007

Sprenkle - CS111

15

## Validating User Input

```
def main():
    #Program mission statement
    print "This program determines your birth year"
    print "given your age and the current year \n"

    age=input("Enter your age: ")
    currentyear=input("Enter the current year: ")

    if age < 0 or age > 115:
        print "Come on: that's not a reasonable age."
    elif currentyear < 0:
        print "You need to have a positive year."
    else:
        birthyear=currentyear - age
        print "You were either born in", birthyear, "or", birthyear-1
```

Mar 28, 2007

Sprenkle - CS111

birthyear.py

16

## Validating User Input

- What happened when the user entered something like "B6"?

Mar 28, 2007

Sprenkle - CS111

17

## Validating User Input

- What happened when the user entered something like "B6"?
  - Threw an **Exception** and the program exited

Python interpreter's message:

```
Enter your age: B6
Traceback (most recent call last):
  File "currentAge.py", line 22, in <module>
    main()
  File "currentAge.py", line 9, in main
    age=input("Enter your age: ")
  File "<string>", line 1, in <module>
NameError: name 'B6' is not defined
```

Mar 28, 2007

Sprenkle - CS111

18

## Handling Exceptions

- Using try/except statements

- Syntax:

```
try:
    <body>
except [<errorType>]:
    <handler>
```

Optional: use this to handle specific error types appropriately

- Example:

```
try:
    age = input("Enter your age: ")
    currentyear = input("Enter the current year: ")
except:
    print "ERROR: Your input was not in the correct form."
    print "Enter integers for your age and the current year"
    return
```

Mar 28, 2007

Sprenkle - CS111

yearborn2.py

19

## Handling Exceptions

- Could put try/catch statements in a loop to make sure user enters valid input

➤ Example: birthyear3.py

- Other types of exceptions

➤ File exceptions:

- File doesn't exist
- Don't have permission to read/write file

Mar 28, 2007

Sprenkle - CS111

file\_handle.py

20

## Broader Issues

Group 1: Vasil, Stuart, Greg  
Group 2: Alex, Clay, Nay, Colin  
Group 3: Joa, Joe, Dave

- One Laptop Per Child
  - An experiment on bringing cheap but educational technology to poor children
- What challenges did OLPC face and how did that affect their design decisions?
- What are some unusual features of the laptop?
- What does this technology mean for better-off countries?
- Is this project worthwhile?

Mar 28, 2007

Sprenkle - CS111

21

## Discussion

Challenge	Design Decision
Lack of power	New, cheap battery; Consumes less power; Alternative power sources: solar power, pull cord
Software bloat	Rewrite code more compactly, efficiently
Environment	Dust proof, drop proof, light
Users: children	Simple user interfaces; tiny keyboard; lightweight; applications keep students interested
Cost	Linux, Python, open-sources tools; cheaper battery; no hard drive; no CD/DVD drive

Mar 28, 2007

Sprenkle - CS111

22