Today's topics

Problem Solving
Pseudocode
Notes from Zachary Dodds' CS 5 course at Harvey Mudd

Upcoming
  > More Python

Reading
  Brookshear, Chapter 5

Problem Solving

Programming is a strenuous exercise in problem solving

● Understand the problem
  > What are its parts? unknown, data, condition
  > Does the problem make sense? Is it feasible?
  > Think about the problem, get a sense of what it needs

● Make a plan
  > Find the connection between givens and result
  > What kind of problem is it? Is it familiar?
  > Think about generalizations, specializations, variants

● Carry out the plan
  > Check each step

● Examine the result
  > Does it make sense?

Are problem solving skills primarily domain-specific?

● You're first told:
  A fortress surrounded by a moat is connected to land by numerous narrow bridges. An attacking army successfully captures the fortress by sending only a few soldiers across each bridge, converging upon it simultaneously.

● Then you're asked:
  A patient has a cancerous tumor. Beams of radiation will destroy the tumor, but in high doses will also destroy healthy tissue surrounding the tumor. How can you use radiation to safely eradicate the tumor?

● Do the skills transfer between subjects? Is a good math problem solver a good computer science problem solver? How about English to Physics?

Programming and Problem Solving

● Latin school movement in the 1600s
  > Teach proper habits of the mind
  > Thorndike’s classic “transfer of training” studies found that learning Latin did not produce strong transfer to other domains

● Programming teaches problem solving movement of late 1900s
  > Seymour Papert, 1980, In learning to program, “powerful intellectual skills are learned in the process
  > The activity of programming computers is fundamentally an exercise in problem solving. The program represents the solution to some problem and the execution behavior of that program becomes a means to judge, unemotionally, the success of the problem solution.

  > Programming is a strenuous exercise in problem solving

  post hoc, ergo propter hoc.
Back of the envelope calculations

http://www.vendian.org/envelope/

- Engineering technique to approximate and check answers
  - Two answers are better than one
  - Quick checks
  - Rules of thumb
  - Practice
- Ad claims that salesperson drove 100,000 miles in a year. True?
- Newspaper article states that a United States quarter dollar coin has “an average life of 30 years.” How can you check that claim?

Why “back of the envelope” estimates?

- Often need to make rapid estimates
  - to eliminate candidate solutions
  - establish feasibility
  - sketch out potential trade-offs
- Most remember key numbers related to their field, not every detail
- Hence we need to estimate
  - which numbers are important
  - values of numbers needed
  - how to perform the calculation
- Emphasis is on “order of magnitude” estimates
  - to nearest factor of 10 (or 2)

Orders of Magnitude

- How far away is home? Is it more like 1, or 10, or 100 miles?
  - Probably do not know exactly
  - Is it approximately “a couple”, or “a few”, or “a lot”
  - Estimate based on powers rather than multiples of 10
- How tall is your dorm? More like 1, 10, 100, 1000 feet?
  - 1 foot tall is like a doll house, so that’s out
  - What do we know that is about 10 feet big? Hmm... People
  - If building is a couple of people high, 10 sounds good.
  - But that means 1000, would be 100 people high, so that’s out
  - So 10 or 100 depending on how many people tall the building is
- Use orders of magnitude as brackets to find reasonable range

Example: How many piano tuners in NYC

- Approximately how many people are in New York City?
  - 10,000,000
- Does every individual own a piano?
  - No
- Reasonable to assert “individuals do not own pianos; families do”?
  - Yes
- About how many families are there in a city of 10 million people?
  - Perhaps there are 2,000,000 families
- Does every family own a piano?
  - No
- Perhaps one out of every five does
  - That would mean there are about 400,000 pianos in NYC
Example: Piano Tuners continued

- How many piano tuners are needed for 400,000 pianos?
  - Some people never get around to tuning their piano
  - Some people tune their piano every month
  - Assume "on the average" every piano gets tuned once a year, then there are 400,000 every year

- How many piano tunings can one piano tuner do?
  - Assume that average piano tuner can tune four pianos a day
  - Assume that there are 200 working days per year
  - That means every tuner can tune about 800 pianos per year

- How many piano tuners are needed in NYC?
  - Number of tuners is approximately 400,000/800 or 500

Example: Piano Tuners summary

- "Back of the Envelope" estimates have
  - Formulas: provide roadmap to upcoming calculations
  - Estimates: brief justification of approximations in formula
  - Calculations: estimates and known facts are use in formula

- Piano Tuner example
  - Formula:
    \[
    \text{# tuners} = \text{# pianos} \times \text{# repairs} / \text{# repairs per day} \times \text{# days}
    \]
  - Estimates
    - \# pianos \approx 400,000 (20% of 2,000,000 families own pianos)
    - \# repairs \approx 1 per piano (some many, some none)
    - \# repairs per day \approx 4
    - \# working days \approx 200 (5 x 50 – vacation, sickness)
  - Calculation
    - \# tuners \approx (400,000 \times 1) / (4 \times 200) = 500

Estimation General Principles

- Recall Einstein's famous advice
  - Everything should be made as simple as possible, but no simpler

- Do not worry about constant factors of 2, \pi, etc.
  - Round to "easy" number or nearest order of magnitude

- Guess numbers you do not know
  - Within bounds of common sense (accuracy increases with experience)

- Adjust geometry, etc., to suit you
  - Assume a cow is spherical if it helps

- Extrapolate from what you do know
  - Use ratios to assume unknown value is similar to known quantity

- Apply a 'plausibility' filter
  - If answer seems unbelievable, it probably is
  - Can usually set range of reasonable values that indicates major mistake (e.g., speed cannot be faster than light!)

What's wrong with this algorithm?

(From back of shampoo bottle)

Directions:
- Wet Hair
- Apply a small amount of shampoo
- Lather
- Rinse
- Repeat
Properties of good algorithms

- Good algorithms must be
  - Correct
  - Complete
  - Precise
  - Unambiguous
- And should be
  - Efficient
  - Simple
  - Contain levels of abstraction

An algorithm is an ordered set of unambiguous, executable steps, defining a terminating process.

Algorithms

- Hand-waving not allowed!
- Specifying algorithms requires you to say what is really involved in making it work.
- Example:
  - How does a computer work?
  - Hand-wave: zeros & ones
  - Real answer: see later part of class.
- You learn to know when you don’t know
  - “I know nothing except the fact of my ignorance.”
  - Socrates, from Diogenes Laertius, Lives of Eminent Philosophers

Describing Algorithms

- Pictures
- Natural language (English)
- Pseudo-code
- Specific high-level programming language

More easily expressed

More precise

Pseudocode

- A shorthand for specifying algorithms
- Leaves out the implementation details
- Leaves in the essence of the algorithm

```plaintext
procedure Greetings
  Count ← 3;
  while (Count < 0) do
    (print the message “Hello” and Count ← Count +1)
```

- What does this algorithm do?
- How many times does it print Hello?
### Picking courses

1. Make a list of courses you want to register for, in order of priority
2. Start with empty schedule. Number of courses = 0.
3. Choose highest priority class on list.
4. If the chosen class is not full and its class time does not conflict with classes already scheduled, then register for the class (2 steps):
   1. Add the class to the schedule
   2. Increment the number of classes scheduled
5. Cross that class off of your list.
6. Repeat steps 3 through 5 until the number of classes scheduled is $\geq 4$, or until all classes have been crossed out.
7. Stop.

### Flowcharts

```
Begin
  Make list of classes you want to take
  Num Classes = 0
  Choose highest priority class on list
  yes
  Is this class full?
  no
  Is there a time conflict?
  yes
  Add the class to your schedule. Increment Num Classes.
  no
  Cross the class off your list.
  yes
  Num Classes $\geq 4$?
  no
  More classes on list?
  yes
  no
End
```

### Programming Primitive Operations

- Assign a value to a variable
- Call a method
- Arithmetic operation
- Comparing two numbers
- Indexing into an array
- Following an object reference
- Returning from a method
Components of Computing Algorithms

Any computing algorithm will have AT MOST five kinds of components:

- Data structures to hold data
- Instructions change data values
- Conditional expressions to make decisions
- Control structures to act on decisions
- Modules to make the algorithm manageable by abstraction, i.e., grouping related components

Game

- 10 coins
  - You and a friend have a stack of 10 coins
  - On each person’s turn, they remove either 1 or 2 coins from the stack
  - The person who removes the last coin wins.
- Can you win?
- 10 coins with a twist
  - 10 coins, can now ALSO place 1 or 2 coins back on the stack
  - Person who removes last coin wins
  - Should you go first or second, and what’s your strategy

Data and lists?

{ 2, 3, 5, 7, 11 }  
Sets

Can all information be represented using lists?

Inside the machine…

What’s happening in python:
\[ x = 41 \]
\[ y = x + 1 \]

What is happening behind the scenes:

"variables as containers"
Python. Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>What is it?</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>3.14</td>
<td>values with a fractional part</td>
</tr>
<tr>
<td>long</td>
<td>10**100</td>
<td>integers &gt; 2147483647</td>
</tr>
<tr>
<td>int</td>
<td>42</td>
<td>integers &lt;= 2147483647</td>
</tr>
<tr>
<td>bool</td>
<td>True, False</td>
<td>the results from a comparison: ==, !=, &lt;, &gt;, &lt;=, =&gt;</td>
</tr>
</tbody>
</table>

Datatypes as genes…

<table>
<thead>
<tr>
<th>Dominant</th>
<th>What will these results be?</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>1.0 / 5</td>
</tr>
<tr>
<td>long</td>
<td>10<strong>100 - 10</strong>100</td>
</tr>
<tr>
<td>int</td>
<td>1 / 5</td>
</tr>
<tr>
<td>bool</td>
<td>41 + True</td>
</tr>
</tbody>
</table>

% the "mod" operator

\( x \% y \) returns the remainder when \( x \) is divided by \( y \)

\[
\begin{align*}
7 \% 3 &= 1 \\
8 \% 3 &= 2 \\
9 \% 3 &= 0 \\
16 \% 7 &= 2
\end{align*}
\]

For what values of \( x \) are these True?

\[
\begin{align*}
x \% 2 &= 0 \\
x \% 2 &= 1 \\
x \% 4 &= 0 \\
x \% 4 &= 1
\end{align*}
\]

What happens on these years?

Python Operators

Precedence: 

() Highest

**

= Lowest

Caution Level:

set equal to =

divide /

remainder %

power **

is equal to ==

as usual 

\[
\begin{align*}
+ & \\
+ & \\
> & \\
< & \\
- & \\
() &
\end{align*}
\]
**Computer Memory**

Random Access Memory (RAM) →

Random Access Memory (RAM) is a long list of memory locations.

- **bit** = 1 "bucket" of charge
- **byte** = 8 bits
- **word** = 4 bytes = 32 bits

Is it really a coincidence that this looks like the string theory picture??

**string functions**

- `str(str(42))` returns '42'
- `len(len('42'))` returns 2
- `+ 'XL' + 'II'` returns 'XLII'
- `* 'VI'*7` returns 'VIVIVIVIVIVIVI'

Given these strings:

\[
\begin{align*}
\text{s1} &= \text{"ha"} \\
\text{s2} &= \text{"t"}
\end{align*}
\]

What are

\[
\text{s1} + \text{s2} \\
2*\text{s1} + \text{s2} + 2*(\text{s1}+\text{s2})
\]

**String surgery**

\[
s = \text{"duke university baby"}
\]

- `s[ ]` indexes into the string, returning a one-character string
- `s[0]` returns 'd'
- `s[6]` returns
- `s[ ]` returns 'e'
- `s[len(s)]` returns

Which index returns 'e'? 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

**More on strings**

- Negative indices count from the back
  - `s[-1]` returns 'y'
  - `s[-11]` returns
  - `s[-0]` returns
  - `s[ : ]` slices the string, returning a substring
  - `s[ : : ]` skip-slices, returning a subsequence

  The third index is the "stride" length

  It defaults to 1
Lists ~ Strings of anything

Square brackets tell python you want a list.

```
L = [ 3.14, [2,40], 'third', 42 ]
```

```
len(L)
```

```
L[0]
```

```
L[0:1]
```

How could you extract from `L` 'hi'

```
message = 'You need parentheses for chemistry !'
```

```
message[::5]
```

```
message[9:15]
```

```
message[0:1]
```

```
message[0][1]
```

```
message[1][0]
```

```
len(pi)
```

```
len(Q)
```

```
len(Q[1])
```

```
len(Q[0][1])
```

```
len(pi)
```

```
len(Q)
```

```
len(Q[1])
```

```
len(Q[0][1])
```

```
abs
```

```
max
```

```
min
```

```
sum
```

```
range
```

```
round
```

```
bool
```

```
float
```

```int
```

```
long
```

```
list
```

```
str
```

```
import math
math.sqrt( 1764 )
```

```
dir(math)
```

```
from math import *
pi
```

```
sin( pi/2 )
```

```
import math
math.sqrt( 1764 )
```

```
dir(math)
```

```
from math import *
pi
```

```
sin( pi/2 )
```

Extra! Mind Muddlers

Part 1

```
len(pi)
```

What are

```
len(Q)
```

```
len(Q[1])
```

What slice of `pi` is `{3,1,4}`

What slice of `pi` is `{3,4,5}`

What are

```
pi[0] * (pi[1] + pi[2])
```

and

```
pi[0] * (pi[1:2] + pi[2:3])
```

Extra! Mind Muddlers

Part 2

```
Q[0]
```

What are

```
Q[0:1]
```

```
Q[0][1]
```

```
Q[1][0]
```

What is `message[9:15]`

What is `message[:5]`

What is `pi[pi[2]]`?

How many nested `pi`'s before `pi[...pi[0]...` produces an error?

"Quiz"
Functioning in Python

# my own function!

def dbl(x):
    """returns double its input, x ""
    return 2*x

Functioning in Python

Some of Python’s baggage...

- **keywords**
  - `def` starts the function
  - `return` stops it immediately and sends back the return value

- **Docstrings**
  - They become part of Python’s built-in help system!
  - With each function be sure to include one that:
    - (1) describes overall what the function does, and
    - (2) explains what the inputs mean/are

- **Comments**
  - They begin with `#`

Functioning in Python

# is it dis-0 or dis-O, anyway?

def undo(s):
    """this "undoes" its string input, s ""
    return 'de' + s

    >>> undo('caf')
    'deaf'
    >>> undo(undo('caf'))

random thoughts

import random

random.choice( range(0,100) )

strings, lists, numbers … all data are fair game

These are the most important: help, dir