Is the Universe a Simulation?

Edward Frenkel

FEB 10, 2014

---

Algorithms

Englishness...
Classifying life
Removing/Sorting and jotto!

HW 4

Hw #4 due Monday, 11:59
Sound Lab!
Several algorithms ...

Office Hrs!

---

Caesar Cipher: $\text{encipher}$

should return the string $s$ with each alphabetic character shifted/wrapped by $n$ places in the alphabet

\[
\text{encipher}(s, n) \rightarrow \text{chr}(\text{ord}(s) + n) \mod 26
\]

\[
\begin{align*}
\text{encipher( 'I <3 Latin', 0 )} & \rightarrow 'I <3 Latin' \\
\text{encipher( 'I <3 Latin', 1 )} & \rightarrow 'J <3 Mbujo' \\
\text{encipher( 'I <3 Latin', 2 )} & \rightarrow 'K <3 Ncvkp' \\
\text{encipher( 'I <3 Latin', 3 )} & \rightarrow 'L <3 Odwlq' \\
\text{encipher( 'I <3 Latin', 4 )} & \rightarrow 'M <3 Pexmr' \\
\text{encipher( 'I <3 Latin', 5 )} & \rightarrow 'N <3 Qfyns' \\
& \vdots \\
\text{encipher( 'I <3 Latin', 25 )} & \rightarrow 'H <3 Kzshm'
\end{align*}
\]

---

Take-away ~ Lab4

```
# Lab4 Caesar Cipher

def flipflop(filename):
    # flipflop swaps the halves of an audio file
    input: filename, the name of the original file
    output: no return value, but this creates the sound file 'out.wav'
    and plays it

flipflop(filenname)

play(filename)

print( "Reading in the sound data...")

sound_data = [0, 0]
read_wave(sound_data)

x = len(samps) // 2
newsamps = [samps[i] for i in range(len(samps) // 2)]

new_sound_data = [newsamps[i] + samps[i] for i in range(len(samps) // 2)]

print( "Computing new sound data...")

write_wave(new_sound_data, "out.wav")

print( "Playing new sound data...")

play(new_sound_data)
```

---

ASCII and Unicode

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
<th>Blw</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100 0000</td>
<td>32</td>
<td>20</td>
<td>blank</td>
<td>@</td>
<td>0100 0000</td>
<td>56</td>
<td>30</td>
</tr>
<tr>
<td>0100 0001</td>
<td>33</td>
<td>21</td>
<td></td>
<td></td>
<td>0100 0001</td>
<td>57</td>
<td>31</td>
</tr>
<tr>
<td>0100 0100</td>
<td>34</td>
<td>22</td>
<td></td>
<td></td>
<td>0100 0101</td>
<td>58</td>
<td>32</td>
</tr>
<tr>
<td>0100 0110</td>
<td>35</td>
<td>23</td>
<td></td>
<td></td>
<td>0100 0111</td>
<td>59</td>
<td>33</td>
</tr>
<tr>
<td>0100 0111</td>
<td>36</td>
<td>24</td>
<td></td>
<td></td>
<td>0100 1000</td>
<td>60</td>
<td>34</td>
</tr>
<tr>
<td>0100 1010</td>
<td>37</td>
<td>25</td>
<td></td>
<td></td>
<td>0100 1011</td>
<td>61</td>
<td>35</td>
</tr>
<tr>
<td>0100 1100</td>
<td>38</td>
<td>26</td>
<td></td>
<td></td>
<td>0100 1101</td>
<td>62</td>
<td>36</td>
</tr>
<tr>
<td>0100 1110</td>
<td>39</td>
<td>27</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>63</td>
<td>37</td>
</tr>
<tr>
<td>0100 1111</td>
<td>40</td>
<td>28</td>
<td></td>
<td></td>
<td>0100 1110</td>
<td>64</td>
<td>38</td>
</tr>
<tr>
<td>0100 1111</td>
<td>41</td>
<td>29</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>65</td>
<td>39</td>
</tr>
<tr>
<td>0100 1111</td>
<td>42</td>
<td>2A</td>
<td></td>
<td></td>
<td>0100 1110</td>
<td>66</td>
<td>3A</td>
</tr>
<tr>
<td>0100 1111</td>
<td>43</td>
<td>2B</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>67</td>
<td>3B</td>
</tr>
<tr>
<td>0100 1111</td>
<td>44</td>
<td>2C</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>68</td>
<td>3C</td>
</tr>
<tr>
<td>0100 1111</td>
<td>45</td>
<td>2D</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>69</td>
<td>3D</td>
</tr>
<tr>
<td>0100 1111</td>
<td>46</td>
<td>2E</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>6A</td>
<td>3E</td>
</tr>
<tr>
<td>0100 1111</td>
<td>47</td>
<td>2F</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>6B</td>
<td>3F</td>
</tr>
<tr>
<td>0100 1111</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td></td>
<td>0100 1111</td>
<td>6C</td>
<td>30</td>
</tr>
<tr>
<td>0100 1111</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td></td>
<td>0100 1111</td>
<td>6D</td>
<td>31</td>
</tr>
<tr>
<td>0100 1111</td>
<td>4A</td>
<td>32</td>
<td>2</td>
<td></td>
<td>0100 1111</td>
<td>6E</td>
<td>32</td>
</tr>
<tr>
<td>0100 1111</td>
<td>4B</td>
<td>33</td>
<td>3</td>
<td></td>
<td>0100 1111</td>
<td>6F</td>
<td>33</td>
</tr>
<tr>
<td>0100 1111</td>
<td>4C</td>
<td>34</td>
<td>4</td>
<td></td>
<td>0100 1111</td>
<td>70</td>
<td>34</td>
</tr>
<tr>
<td>0100 1111</td>
<td>4D</td>
<td>35</td>
<td>5</td>
<td></td>
<td>0100 1111</td>
<td>71</td>
<td>35</td>
</tr>
<tr>
<td>0100 1111</td>
<td>4E</td>
<td>36</td>
<td>6</td>
<td></td>
<td>0100 1111</td>
<td>72</td>
<td>36</td>
</tr>
<tr>
<td>0100 1111</td>
<td>4F</td>
<td>37</td>
<td>7</td>
<td></td>
<td>0100 1111</td>
<td>73</td>
<td>37</td>
</tr>
<tr>
<td>0100 1111</td>
<td>50</td>
<td>38</td>
<td>8</td>
<td></td>
<td>0100 1111</td>
<td>74</td>
<td>38</td>
</tr>
<tr>
<td>0100 1111</td>
<td>51</td>
<td>39</td>
<td>9</td>
<td></td>
<td>0100 1111</td>
<td>75</td>
<td>39</td>
</tr>
<tr>
<td>0100 1111</td>
<td>52</td>
<td>3A</td>
<td>:</td>
<td></td>
<td>0100 1111</td>
<td>76</td>
<td>3A</td>
</tr>
<tr>
<td>0100 1111</td>
<td>53</td>
<td>3B</td>
<td>;</td>
<td></td>
<td>0100 1111</td>
<td>77</td>
<td>3B</td>
</tr>
<tr>
<td>0100 1111</td>
<td>54</td>
<td>3C</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>78</td>
<td>3C</td>
</tr>
<tr>
<td>0100 1111</td>
<td>55</td>
<td>3D</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>79</td>
<td>3D</td>
</tr>
<tr>
<td>0100 1111</td>
<td>56</td>
<td>3E</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>80</td>
<td>3E</td>
</tr>
<tr>
<td>0100 1111</td>
<td>57</td>
<td>3F</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>81</td>
<td>3F</td>
</tr>
<tr>
<td>0100 1111</td>
<td>58</td>
<td>40</td>
<td></td>
<td></td>
<td>0100 1111</td>
<td>82</td>
<td>40</td>
</tr>
</tbody>
</table>

This is why 'CS' < 'clear'!
def rot13(c):
    """ rotates c by 13 chars, "wrapping" as needed 
    NON-LETTERS don't change! 
    """
    if 'a' <= c <= 'z':
        if ord(c)+13 <= ord('z'):
            return chr(ord(c)+13 )
        else:
            return chr(ord(c)+13 -26 )
    elif 'A' <= c <= 'Z':
        # upper-case test!
    else:
        # Extra: How would you rotate n places, instead of 13?
        return c

(0) What do these tests do?
(1) What code will "wrap" to the alphabet's other side?
(2) How will upper case change? Try noting only the code differences...
(3) What if c is not a letter at all?

Extra: How would you rotate n places, instead of 13?
Measuring Englishness

Very English-y
"Call me Ishmael." "Attack at dawn!"
"rainbow, table, candle"
"quadruplicity drinks procrastination"
"Hold the newsreader’s nose squarely, waiter, or friendly milk will countermand my trousers."

Not English-y
"Yow! Legally-imposed CULTURE-reduction is CABBAGE-BRAINED!"
"the gostak distims the doshes"
"hension, framble, bardle"
"jufict, stofwus, lictpub"
"epadxo, nojarpn, gdxokpw"
"h o q dedqBzdrzqrzkzc"

All of these sound good to me!

Decipher?

Strategies?

Algorithms?

Score them all

"Englishness " score ~ the #of vowels

Design...

Code?

Algorithm Design...

remAll(e,L)
remove all e’s from L
Design...

**Top-down design**
- Visualize
- Split into parts
- Build each part
- Combine
- Test

**Design ~ code**

```python
def remAll(e, L):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])
```

If there are no elements or characters in L, we're done — return L itself!

**remAll insight**

```python
def remAll(e, L):
    """ removes all e's from L ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])
```

**other rem examples...**

<table>
<thead>
<tr>
<th>Function</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>remAll(8, [7,8,9,8])</code></td>
<td><code>[7,9]</code></td>
<td><code>remAll</code></td>
</tr>
<tr>
<td><code>remOne('d', 'coded')</code></td>
<td><code>'coed'</code></td>
<td><code>remOne</code></td>
</tr>
<tr>
<td><code>remOne(8, [7,8,9,8])</code></td>
<td><code>[7,9,8]</code></td>
<td><code>remOne</code></td>
</tr>
<tr>
<td><code>remUpto(8,[7,8,9,8])</code></td>
<td><code>[9,8]</code></td>
<td><code>remUpto</code></td>
</tr>
<tr>
<td><code>remUpto('d','coded')</code></td>
<td><code>'ed'</code></td>
<td><code>remUpto</code></td>
</tr>
</tbody>
</table>

sharpening our model for where + how actions happen...

remAll(8, [7,8,9,8]) ➞ [7,9]
remAll(42,[5,7,42,8,42]) ➞ [5,7,8]
remAll('q','qaqqlqqiqqiqeqqns') ➞ 'aliiiens'
remAll(8, [7,8,9,8]) ➞ [7,9]  remAll
remAll('d', 'coded') ➞ 'coed'  remAll
remOne(8, [7,8,9,8]) ➞ [7,9,8]  remOne
remOne('d', 'coded') ➞ 'coed'  remOne
remUpto(8,[7,8,9,8]) ➞ [9,8]  remUpto
remUpto('d','coded') ➞ 'ed'  remUpto
Subsequences

```python
def subseq(s, sbig):
    s is the subsequence to find (or not)
    sbig is the bigger string in which we are looking for s

subseq('','cataga') → True
subseq('ctg','cataga') → True
subseq('ctg','tacgta') → False
subseq('aliens','always frighten dragons') → True
subseq('trogdor','that dragon is gone for good') → False
```

Why Are these True? or False?

Quiz
def remAll(e, L):
    """ returns seq. L with all e's removed ""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])

remAll(8,[7,8,9,8]) → [7,9,8]
remAll('d','coded') → 'coed'

Algorithm design

1. Change remAll so that it removes only one e from L (We could call it remOne).

2. Write the other cases needed for subseq…

3. Write the other cases needed for subseq…

from remAll to remOne

```python
def remOne(e, L):
    """returns seq. L with one e removed""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remOne(e, L[1:])
    else:
        return L[1:]

remOne(8,[7,8,9,8]) → [7,9,8]
remOne('d','coded') → 'coed'
```
Subseq ~ trying (coding) it out...

```python
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
    False otherwise. Both are strings. 
    """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    elif s[0] == sbig[0]:
        return subseq( s[1:], sbig[1:] )
    else:
        return subseq( s[0:], sbig[1:] )
```

Base case(s)
Recursive step(s)

Use it!
Lose it!

Subseq ~ thinking it out...

```python
subseq( 'ctg', 'tacgta' )
```

Top-down design

Use it!
Lose it!

What is a small (initial) piece of the problem?
How would we describe it in terms of the inputs?

Where are the useit and loseit here?

Subseq ~ trying (coding) it out...

```python
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
    False otherwise. Both are strings. 
    """
    if s == '':
        return True
    elif s[0] not in sbig:
        return False
    elif s[0] == sbig[0]:
        return subseq( s[1:], sbig[1:] )
    else:
        return subseq( s[0:], sbig[1:] )
```

Design ~ (code)

That's it. Algorithmic expression ~ it's what CSers (think they) do.
What's the problem?!

Top-down design
- Visualize
- Split into parts
- Build each part
- Combine
- Test

Which one of these steps is the most important?

hw4pr2: use it or lose it

Longest Common Subsequence
- LCS(S, T)

Jotto Score counting
- jscore(s1, s2)

binary list and general list sorting
- blsort(L), gensort(L)

exact_change making
- exact_change(t, L)

hw4pr2: use it or lose it

Longest Common Subsequence
- LCS(S, T)

'Human'

'CGCTGAGCTAGGCA...'

'chimpANZEE'

'ATCCTAGGTAACTG...'

Why LCS?

Screenshot from the ClustalX multiple subsequence alignment tool...

Algorithmic challenge: How to find the best common subsequences among these very big genome strings?!!
What was gained (or lost) here?

Phylogeny

Subsequences @ 5Cs

host: figs

parasites: wasps

matching two phylogenies together

together!

Jane's source data: 100s of species, 6 continents ...

Largest co-phylogeny ever computed (in 2012)

also in hw4pr2: Jotto!

a word-guessing game...

jscore( S, T )
These are two cute 'robot' 'otter'

\[
\text{jscore}('robot', 'otter') \rightarrow \text{jscore}(S, T) \quad \text{in general...}
\]

\[
\text{sort}(L) \quad \text{should return the Lonest Common Subsequence of strings } S \text{ and } T
\]

\[
\text{exact}_\text{change}(42, [25,30,2,5]) \rightarrow \text{exact}_\text{change}(42, [25,30,2,15]) \rightarrow
\]

\[
\text{decipher('Weet bksa ed xecumeha 4!')}
\]

\[
\text{Brainstorm algorithms for these problems -- what helper functions?? might help for each?}
\]

\[
\text{also in hw3pr2: } \text{sort + exact}_\text{change}
\]

\[
\text{sort}([42,5,7]) \rightarrow \text{sort}([42,7]) \rightarrow \text{sort}([42]) \rightarrow
\]

\[
\text{return an ascending list returns True or False}
\]

\[
\text{exact}_\text{change}(42, [25,30,2,5]) \rightarrow \text{exact}_\text{change}(42, [25,30,2,15]) \rightarrow
\]

\[
\text{exact}_\text{change}(20, [16,3,2,17]) \rightarrow
\]
def remAll(e, L):
    """removes all e's from L""
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return remAll(e, L[1:])

def subseq(s, sbig):
    """returns True if s is a subseq. of sbig, False otherwise. Both are strings."
    if s == '':
        return True
    else:
        return subseq(s[1:], sbig) and s[0] in sbig

Challenge... Write the other cases needed for subseq...

Hint: you'll need 3-4 cases total for subseq.

remOne(8, [7, 8, 9, 8])  ➞  [7, 9, 8]

remUpto('d', 'coded')  ➞  'ed'

Hint: In both cases, what's needed is *mostly crossing stuff out! What stuff?*

Hint: remove one thing for remOne!

Hint: remove one *more* thing for remUpto!

Algorithm design

1. Change remAll so that it removes only one e from L. (We could call it remOne.)
2. Make more changes to remAll so that it removes all of the elements up to and including the first e in L. (We could call it remUpto.)
3. Write the other cases needed for subseq...

cs5 hrs last week

subseq('alg', 'magical')
subseq('alg', 'twasbrillig')
subseq('alg', 'magical')
subseq('alg', 'twasbrillig')

True

Challenge...