More Recursion

[Andersen, Gries, Lee, Marschner, Van Loan, White]
Announcements

• We can’t check off labs in professor office hours
• Reading for next week: Chapters 15 and 16
Announcements: A3

• **Due**: Thursday, March 30\textsuperscript{th}, 11:59pm

• trigram\_generation: “REQUIREMNET [sic]: first, randomly pick a starting bigram "w1 w2".”

• This means, “pick “w1 w2” randomly from the sample text, just like you picked a unigram from the text in bigram\_generation."
Recall: Divide and Conquer

Goal: Solve problem P on a piece of data

data

Idea: Split data into two parts and solve problem

data 1  data 2

Solve Problem P  Solve Problem P

Combine Answer!
def reverse(s):
    """Returns: reverse of s""
    # 1. Handle small data

    # 2. Break into two parts

    # 3. Combine the result
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"
    # 1. Handle small data
    # 2. Break into two parts
    # 3. Combine the result
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"
    # 1. Handle small data
    # 2. Break into two parts
    # 3. Combine the result
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string""
    # 1. Handle small data
    # 2. Break into two parts
    # 3. Combine the result
    return A: left + right  B: right + left  C: left  D: right

3/23/17 More Recursion
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"
    # 1. Handle small data
    # 2. Break into two parts
    left = reverse(s[0])
    right = reverse(s[1:])
    # 3. Combine the result
    return right+left
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string""
    # 1. Handle small data
    left = reverse(s[0])
    right = reverse(s[1:])
    # 2. Break into two parts
    if s == "":
        return s
    if len(s) <= 2:
        return s
    if len(s) <= 1:
        return s
    # 3. Combine the result
    return right+left

Note: This question was problematic as presented in lecture, so it has been changed a bit.

CORRECT
A: if s == "":
    return s
B: if len(s) <= 2:
    return s
C: if len(s) <= 1:
    return s

D: Either A or C would work
E: A, B, and C would all work
Note: This question was problematic as presented in lecture, so it has been changed a bit.

```python
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string""
    # 1. Handle small data
    if s == "": return s
    A: if len(s) <= 2: return s
    B: if len(s) <= 1: return s
    # 2. Break into two parts
    left = s[0]
    right = reverse(s[1:])
    # 3. Combine the result
    return right + left
```

CORRECT

D: Either A or C would work

E: A, B, and C would all work
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"
    # 1. Handle small data
    if len(s) <= 1:
        return s

    # 2. Break into two parts
    left = s[0]
    right = reverse(s[1:])

    # 3. Combine the result
    return right+left

Base Case

Recursive Case
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"
    # 1. Handle small data
    if len(s) <= 1:
        return s
    # 2. Break into two parts
    left = reverse(s[:len(s)-1])
    right = reverse(s[len(s)-1])
    # 3. Combine the result
    return right+left

A: YES

B: NO

Does this work?

CORRECT
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"
    if len(s) <= 1:
        return s
    left = reverse(s[:2])
    right = reverse(s[2:])
    return right+left

3/23/17 More Recursion 14

Does this work?
A: YES
CORRECT
B: NO
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string""
    # 1. Handle small data
    if len(s) <= 1:
        return s
    # 2. Break into two parts
    left = reverse(s[:2])
    right = reverse(s[2:])
    # 3. Combine the result
    return right + left

Uh oh. Not proceeding.
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string""
    # 1. Handle small data
    if len(s) <= 1:
        return s
    if len(s) == 2:
        return s[1] + s[0]
    # 2. Break into two parts
    left = reverse(s[:2])
    right = reverse(s[2:]
    # 3. Combine the result
    return right+left

Does this work?

CORRECT [A: YES  B: NO]
def reverse(s):
    """Returns: reverse of s
    Precondition: s a string"""
    # 1. Handle small data
    if len(s) <= 1:
        return s

    # 2. Break into two parts
    half   = len(s)/2
    left   = reverse(s[:half])
    right  = reverse(s[half:]),

    # 3. Combine the result
    return right+left

3/23/17 More Recursion 17

A: YES
B: NO

Does this work?
Alternate Implementation

reverse(s[:half])

reverse(s[::half])

reverse(s[:half])

reverse(s[::half])

reverse(s[:half])

reverse(s[::half])

reverse(s[:half])

reverse(s[::half])

Half = 3

Half = 1

Half = 1

Half = 1

Half = 1

Half = 1
Alternate Implementation

reverse(s[:half])

reverse(s[half:])

reverse(s[:half])

reverse(s[half:])

reverse(s[:half])

reverse(s[half:])

reverse(s[:half])

reverse(s[half:])

reverse(s[:half])

reverse(s[half:])

reverse(s[:half])

reverse(s[half:])

reverse(s[:half])

reverse(s[half:])
Example: Palindromes

- Example:

  AMANAPLANACANALPANAMA

- Can we define recursively?
Example: Palindromes

- String with $\geq 2$ characters is a palindrome if:
  - its first and last characters are equal, and
  - the rest of the characters form a palindrome

- Example:
  - AMANAPLANACANALPANAMA

- Implement: `def ispalindrome(s):

  """Returns: True if s is a palindrome"""
Example: Palindromes

- String with ≥ 2 characters is a palindrome if:
  - its first and last characters are equal, and
  - the rest of the characters form a palindrome

```python
def ispalindrome(s):
    """Returns: True if s is a palindrome""
    if len(s) < 2:
        return True
    ends = s[0] == s[-1]
    middle = ispalindrome(s[1:-1])
    return ends and middle
```

Recursive case

Base case

Recursive Definition
Recursion and Objects

- Class Person (person.py)
  - Objects have 3 attributes
    - **name**: String
    - **parent1**: Person (or None)
    - **parent2**: Person (or None)
  - Represents the “family tree”
    - Goes as far back as known
    - Attributes `parent1` and `parent2` are None if not known
- **Constructor**: Person(name,p1,p2)
  - Or Person(n) if no parents known
def num_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""
    # 1. Handle small data.
    # No parent1 or parent2
    #(no ancestors)

    # 2. Break into two parts
    # Has parent1 or parent2
    # Count ancestors of each one
    #(plus parent1, parent2 themselves)

    # 3. Combine the result

11 ancestors
def num_ancestors(p):
    
    # 1. Handle small data.
    if p.parent1 == None and p.parent2 == None:
        return 0

    # 2. Break into two parts
    parent1s = 0
    if p.parent1 != None:
        parent1s = 1 + num_ancestors(p.parent1)
    parent2s = 0
    if p.parent2 != None:
        parent2s = 1 + num_ancestors(p.parent2)

    # 3. Combine the result
    return parent1s + parent2s
def num_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""
    # 1. Handle small data.
    if p.parent1 == None and p.parent2 == None:
        return 0
    # 2. Break into two parts
    parent1s = 0
    if p.parent1 != None:
        parent1s = 1+num_ancestors(p.parent1s)
    parent2s = 0
    if p.parent2 != None:
        parent2s = 1+num_ancestors(p.parent2s)
    # 3. Combine the result
    return parent1s+parent2s

Recursion and Objects

We don’t actually need this.
It is handled by the conditionals in #2.
def all_ancestors(p):
    """Returns: list of all ancestors of p"""
    # 1. Handle small data.
    # 2. Break into parts.
    # 3. Combine answer.