Discussion 4: Tree Recursion, Lists

Gabe Classon's CS 61A discussion

9:30–11:00 a.m. Friday, February 17, 2023

Question of the day

What's a talent you wish you had?

Announcements

- We won't get to everything... and that's OK
- UAW 2865 is bargaining to save EECS, and we need your input
  - https://tinyurl.com/UAW-EECS-survey
  - Will email to you after class

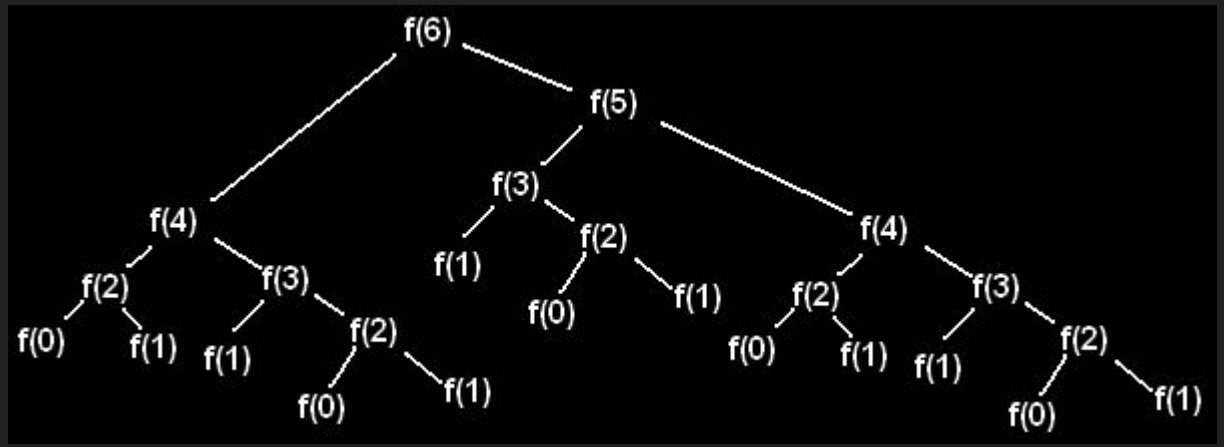
Tree recursion

Tree recursion

- Instead of one recursive call, there is more than one

Virahanka-Fibonacci

```
def virfib(n):
    if n == 0 or n == 1:
        return n
    return virfib(n - 1) + virfib(n - 2)
```



Tree recursive function design

- Base case: What's the simplest version of this problem?
- Recursive call: How can I break this down into multiple smaller versions of the same problem?
  - Consider the outcome of a choice.
  -
- Solving the larger problem: How can I combine my multiple solutions into a larger solution?

Q1: Count Stair Ways

Imagine that you want to go up a flight of stairs that has `n` steps, where `n` is a positive integer. You can either take `1` or `2` steps each time. How many different ways can you go up this flight of stairs? In this question, you'll write a function `count_stair_ways` that solves this problem.

```
def count_stair_ways(n):
    """Returns the number of ways to climb up a flight of
    n stairs, moving either 1 step or 2 steps at a time.
    >>> count_stair_ways(4)
    5
    """
    "*** YOUR CODE HERE ***"
```

Q1: Count Stair Ways (answer)

```
def count_stair_ways(n):
    if n == 1:
        return 1
    elif n == 2:
        return 2
    return count_stair_ways(n-1) + count_stair_ways(n-2)

def count_stair_ways(n):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    return count_stair_ways(n-1) + count_stair_ways(n-2)
```

Q1: Count Stair Ways (answer)

```
def count_stair_ways(n):
    if n == 0 or 1:
        return 1
    return count_stair_ways(n-1) + count_stair_ways(n-2)
```

This is the Virahanka-Fibonacci sequence (!)

Q2: Count K

Consider a special version of the `count_stair_ways` problem, where instead of taking `1` or `2` steps, we are able to take up to and including `k` steps at a time. Write a function `count_k` that figures out the number of paths for this scenario. Assume `n` and `k` are positive.

```
def count_k(n, k):
    """ Counts the number of paths up a flight of n
stairs
    when taking up to and including k steps at a
time.
    >>> count_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1
    4
    >>> count_k(4, 4)
    8
    >>> count_k(10, 3)
    274
    >>> count_k(300, 1) # Only one step at a time
    1
    """
    "*** YOUR CODE HERE ***"
```

Q2: Count K (answer)

```
def count_k(n, k):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    else:
        total = 0
        i = 1
        while i <= k:
            total += count_k(n - i, k)
            i += 1
        return total
```

Lists

List

A *list* stores multiple elements.

```
>>> list_of_ints = [1, 2, 3, 4]
```

```
>>> list_of_bools = [True, True, False, False]
```

```
>>> nested_lists = [1, [2, 3], [4, [5]]]
```

Item selection

Lists have item selection and length.

```
>>> lst = [6, 5, 4, 3, 2, 1, 0]
>>> lst[0]
6
>>> lst[3]
3
>>> lst[-1]
0
>>> len(lst)
7
```

`in` keyword

`in` tests for list inclusion.

```
>>> 2 in [3, 4, 2]
True
>>> 5 in ["5", 7]
False
```

Truthiness and falsiness

Empty lists are falsy; nonempty lists are truthy.

What do the following output?

```
if []:
    print(1)

if [[]]:
    print(2)
2

if [0]:
    print(3)
3
```

Q3: WWPD: Lists

```
>>> a = [1, 5, 4, [2, 3], 3]
>>> print(a[0], a[-1])
1 3
>>> len(a)
5
>>> 2 in a
False
>>> a[3][0]
2
```

Iterating through a list

You can use lists in `for` loops:

```
for x in ["a", 3, True]:
    print(x)
```

```
a
3
True
```

Iterating through a range of numbers

You can also iterate through numbers

```
for x in range(3):
    print(x)
```

```
0
1
2
```

```
lst = ['a', 'b', 'c']
for x in range(len(lst)):
    print(x)
```

```
0
1
2
```

Iterating through a range of numbers

```
lst = ['a', 'b', 'c']

for x in range(len(lst)):

    print(lst[x])
```

```
a

b

c
```

List comprehensions

A fast way to iterate through a sequence to create a new list.

```
lst = [<expression> for <element> in <seq> if <conditional>]
```

is equivalent to:

```
lst = []
for <element> in <seq>
    if <conditional>:
        lst += [<expression>]
```

Example

What does this evaluate to?

```
[x for x in range(10)]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Q4: Even weighted

Write a function that takes a list `s` and returns a new list that keeps only the even-indexed elements of `s` and multiplies them by their corresponding index. First, implement `even_weighted` with a for loop. Then do it with a list comprehension.

```
def even_weighted(s):
    """
    >>> x = [1, 2, 3, 4, 5, 6]
    >>> even_weighted(x)
    [0, 6, 20]
    """
```

Q4: Even weighted (answer)

Write a function that takes a list `s` and returns a new list that keeps only the even-indexed elements of `s` and multiplies them by their corresponding index. First, implement `even_weighted` with a for loop.

```
def even_weighted_loop(s):
    result = []
    for i in range(len(s)):
        if i % 2 == 0:
            result = result + [i * s[i]]
    return result
```

Q4: Even weighted (answer)

Write a function that takes a list `s` and returns a new list that keeps only the even-indexed elements of `s` and multiplies them by their corresponding index. Then do it with a list comprehension.

```
def even_weighted_comprehension(s):
    return [i * s[i] for i in range(len(s)) if i % 2 == 0]
```

List arithmetic

Adding two lists concatenates them:

```
>>> [4, 9] + [[3], 'a', 'b']
[4, 9, [3], 'a', 'b']
```

Multiplying a list by a number *n* concatenates it to itself *n* times:

```
>>> 3 * [1, 2]
[1, 2, 1, 2, 1, 2]
```

Slicing

A cool way to create a (possibly modified) copy of `list`

For a list lst, syntax is `lst[<start index>:<end index>:<step size>]`

Includes elements starting at `<start index>` and up to but not including the `<end index>` taking steps of size `<step size>`.

If values not provided:

- `<start index>` is `0`
- `<end index>` is `len(s)`
- `<step size>` is `1`

Slicing examples

```
>>> lst[:3]   # Start index defaults to 0
[6, 5, 4]
>>> lst[3:]   # End index defaults to len(lst)
[3, 2, 1, 0]
>>> lst[::-1]   # Make a reversed copy of the entire list
[0, 1, 2, 3, 4, 5, 6]
>>> lst[::2]  # Skip every other; step size defaults to 1
otherwise
[6, 4, 2, 0]
```

Q5: Max product

Write a function that takes in a list and returns the maximum product that can be formed using nonconsecutive elements of the list. The input list will contain only numbers greater than or equal to 1.

```
def max_product(s):
    """Return the maximum product that can be formed using
    non-consecutive elements of s.
    >>> max_product([10,3,1,9,2]) # 10 * 9
    90
    >>> max_product([5,10,5,10,5]) # 5 * 5 * 5
    125
    >>> max_product([])
    1
    """
```

Q5: Max product (answer)

Write a function that takes in a list and returns the maximum product that can be formed using nonconsecutive elements of the list. The input list will contain only numbers greater than or equal to 1.

```
def max_product(s):
    if not s:
        return 1
    else:
        return max(max_product(s[1:]),
                   s[0] * max_product(s[2:]))
```

Dictionaries

An unordered collection of key-value pairs. Keys are inputs, values are outputs.

```
>>> pokemon = {'pikachu': 25, 'dragonair': 148, 'mew': 151}
>>> pokemon['pikachu']
25
>>> pokemon['jolteon'] = 135
>>> pokemon
{'jolteon': 135, 'pikachu': 25, 'dragonair': 148, 'mew': 151}
>>> pokemon['ditto'] = 25
>>> pokemon
{'jolteon': 135, 'pikachu': 25, 'dragonair': 148,'ditto': 25, 'mew': 151}
```

Dictionaries

- Dictionaries have length
  `len({1: 2, 3: 4})` → `2`
- Dictionaries (keys) can be iterated through

```
for i in {1: 2, 3: 4}:
    print(i)
1
3
```

- `in` tests for dictionary (key) inclusion
  `2 in {1: 2, 3: 4}` → `False`
  `>>> 1 in {1: 2, 3: 4}` → `True`

Are dictionaries ordered?

No.

Q6: WWPD: Dictionaries

```
>>> pokemon = {'pikachu': 25, 'dragonair': 148}
>>> pokemon
{'pikachu': 25, 'dragonair': 148}
>>> 'mewtwo' in pokemon
False
>>> len(pokemon)
2
```

Q6: WWPD: Dictionaries

```
>>> pokemon['mew'] = pokemon['pikachu']
>>> pokemon[25] = 'pikachu'
>>> pokemon
{'pikachu': 25, 'dragonair': 148, 'mew': 25, 25: 'pikachu'}
>>> pokemon['mewtwo'] = pokemon['mew'] * 2
>>> pokemon
{'pikachu': 25, 'dragonair': 148, 'mew': 25, 25: 'pikachu', 'mewtwo': 50}
>>> pokemon[['firetype', 'flying']] = 146
Error: unhashable type
```

Attendance

Fill out gabeclasson.com/attend

(or go to the section website gabeclasson.com/cs61a)

The secret word is

corpus

A collection of writings, often on a specific topic, of a specific genre, from a specific demographic or a particular author, etc.