Lab 5

Recursivity

Sup'Biotech 3

Python

Pierre Parutto

October 19, 2016

Preamble

Document Property

Authors	Pierre Parutto
Version	1.0
Number of pages	9

Contact

Contact the assistant team at: supbiotech-bioinfo-bt3@googlegroups.com

Copyright

Contents

1 Introduction

In this sixth lab, we will manipulate recursivity.

2 Warm-up

2.1 Dot Product

I remind you the dot product formula between two vectors $X = X_1, \ldots, X_N$ and $Y = Y_1, \ldots, Y_N$:

$$X \cdot Y = \sum_{i=1}^{N} X_i \times Y_i$$

1. Formulate the problem recursively.
2. Write a recursive function `dot_prod_rec(X: list, Y: list) -> int` that computes the dot product between the two vectors `X` and `Y`. We consider that `X` and `Y` have the same size.

Example

```
>>> dot_prod_rec([1,2], [3,4])
11
>>> dot_prod_rec([4,3], [3,4])
24
```

Correction:

```
def dot_prod_rec(X,Y):
    if len(X) == 0:
        return 0
    return X[0] * Y[0] + dot_prod_rec(X[1:], Y[1:])
```

3 Recursivity With Sequences

3.1 1 Recursive Case, 1 Stop Case - Pow

The power a^b can be computed as:

$$a^b = \begin{cases} a \times a^{b-1} & \text{if } b > 0 \\ 1 & \text{otherwise} \end{cases}$$

Write a recursive function `pow(a: int, b: int) -> int` that returns the value a^b.

Example

```
>>> pow(2,5)
32
>>> pow(4, 8)
65536
```

Correction:

```
def pow(a, b):
    if b == 0:
        return 1
    return a * pow(a, b-1)
```

3.2 1 Recursive Case, 1 Stop Case - A Sequence

Write a recursive function `seq(n: int) -> int` that returns the value u_n of the following sequence:

$$u_n = \begin{cases} 3 \times u_{n-1} - 1 & n = 0 \\ 0 & \text{otherwise} \end{cases}$$

Example

```
>>> seq(1)
-1
>>> seq(3)
-13
```

Correction:

```
def seq(n):
        if n == 0:
                return 0
        return 3 * seq(n - 1) - 1
```

3.3 1 Recursive Case, 2 Stop Cases - Fibonacci Sequence

Write a recursive function `fibo(n: int) -> int` that returns the n^{th} value of the Fibonacci sequence:

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Example

```
>>> fibo(2)
1
>>> fibo(11)
89
```

Correction:

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

3.4 2 Recursive Cases, 1 Stop Case - Fast Pow

Write a recursive function `fast_pow(a: int, b: int) -> int` that returns a^b using the following formula:

$$a^b = \begin{cases} a \times (a^2)^{\frac{b-1}{2}} & \text{if } b \text{ is odd} \\ (a^2)^{\frac{b}{2}} & \text{if } b \text{ is even} \\ 1 & \text{if } b = 0 \end{cases}$$

Example

```
>>> fast_pow(2, 3)
8
>>> fast_pow(5, 6)
15625
```

Correction:

```
def fast_pow(a, b):
        if b == 0:
                return 1
        if b % 2 == 1:
                return a * fast_pow(a * a, (b - 1) / 2)
        return fast_pow(a * a, b / 2)
```

4 Recurisivity With Lists

4.1 Number Of Elements

We want to count the numbers of elements in a list.

1. Formulate the problem recursively.
2. Write a recursive function `count(l: list) -> int` that returns the number of elements in the list `l`.

Example

```
>>> count([1,2,3])
3
>>> count([1,5,8,6])
4
```

Correction:

```
def count(l):
	if l == []:
		return 0
	return 1 + count(l[1:])
```

4.2 Sum Of Elements

We want to sum all the elements in a list.

1. Write the problem recursively.
2. Write a recursive function `sum_list(l: list) -> float` that returns the sum of the elements in the list `l`.

Example

```
>>> sum_list([1,2,3])
6
>>> sum_list([-1.5, 0.8, 1.2])
0.5
```

Correction:

```
def sum_list(l):
	if l == []:
		return 0
	return l[0] + sum_list(l[1:])
```

4.3 Filtering

We want to remove all the odds elements from a list.

1. Formulate the problem recursively.
2. Write a recursive function `filter_odd(l: list) -> list` that returns the list `l` **without** all the odd elements.

Example

```
>>> filter_odd([1, 2, 3])
[2]
>>> filter_odd([7, 9, 11])
[]
```

Correction:

```
def filter_odd(l):
        if l == []:
                return []
        if l[0] % 2 == 1:
                return filter_odd(l[1:])
        return [l[0]] + filter_odd(l[1:])
```

4.4 Inverting A List

We want to invert the elements of a list.

1. Formulate the problem recursively.
2. Write a recursive function `invert(l: list) -> float` that return the list `l` inverted (the first element becomes the last, etc).

Example

```
>>> invert([1, 2])
[2, 1]
>>> invert([5, 4, 3, 2, 1])
[1, 2, 3, 4, 5]
```

Correction:

```
def invert(l):
        if l == []:
                return []
        return invert(l[1:]) + [l[0]]
```

5 Recursivity With Strings

5.1 Distance

A measure of the distance between two strings is defined as:

$$D(s_1, s_2) = \begin{cases} |s_1| & \text{if } s_2 \text{ is empty} \\ |s_2| & \text{if } s_1 \text{ is empty} \\ D(\text{suffix}(s_1), \text{suffix}(s_2)) & \text{if } s_1[0] = s_2[0] \\ 1 + D(\text{suffix}(s_1), \text{suffix}(s_2)) & \text{if } s_1[0] \neq s_2[0] \end{cases}$$

where $|x|$ is the length of the sequence x and $\text{suffix}(x)$ is the sequence x **without the first character**.

Write a recursive function `dist_str(s1: str, s2: str) -> int` that returns the distance D between the two sequences `s1` and `s2`. The two sequences can have different lengths.

Example

```
>>> dist_str("AAAAA", "")
5
>>> dist_str("AAAAA", "ATGC")
4
```

Correction:

```
def dist_str(s1, s2):
    if len(s1) == 0:
        return len(s2)
    if len(s2) == 0:
        return len(s1)
    if s1[0] == s2[0]:
        return dist_str(s1[1:], s2[1:])
    return 1 + dist_str(s1[1:], s2[1:])
```

5.2 Levenshtein Distance

The Levenshtein distance (also called edit distance) between two strings is defined as:

$$L(s_1, s_2) = \begin{cases} L(\text{suffix}(s_1), \text{suffix}(s_2)) & \text{if } s_1[0] = s_2[0] \\ \min \begin{cases} 1 + L(\text{suffix}(s_1), \text{suffix}(s_2)) \\ 1 + L(s_1, \text{suffix}(s_2)) \\ 1 + L(\text{suffix}(s_1), s_2) \end{cases} & \text{if } s_1[0] \neq s_2[0] \\ |s_1| & \text{if } |s_2| = 0 \\ |s_2| & \text{if } |s_1| = 0 \end{cases}$$

where $|x|$ is the length of the sequence x and $\text{suffix}(x)$ is the sequence x **without the first character**.

Write a recursive function `levenshtein(s1, s2)` that returns the Levenshtein distance between the two sequences `s1` and `s2`.

Example

```
>>> levenshtein("ATTGT", "")
5
>>> levenshtein("ATTGT", "AT")
3
>>> levenshtein("AATTTGTC", "ATTGT")
3
```

Correction:

```
def levenshtein(s1, s2):
    if len(s1) == 0:
        return len(s2)
    if len(s2) == 0:
        return len(s1)
    if s1[0] == s2[0]:
        return levenshtein(s1[1:], s2[1:])
    return 1 + min(levenshtein(s1[1:], s2[1:]),
                   levenshtein(s1[1:],s2),
                   levenshtein(s1, s2[1:]))
```