BCM0505-15 - Processamento da Informação

Feb 10, 2020

Laboratório L01 – 21/04

Instruções

- Este é o primeiro laboratório do período de ECE.
- Como hoje é feriado, a resolução oficial do ECE estabelece que não teremos atividade síncrona (plantão de dúvidas).
- Todos os exercícios nesta página têm o propósito de revisão dos conceitos de variáveis, laços (`for` e `while`), seleções (`if`, `elif`, `else`), funções (`def` e `return`) e listas.
- Membros das turmas NA1 e NB{4,5} deverão submeter códigos em Python para todos os exercícios nesta página em um único arquivo `.py` via Tidia até 28/04 às 19h – sob a entrada **E5.**

Exercícios

- **(01)** *Inteiros Balanceados I.*
  Um inteiro positivo n com k dígitos decimais é *balanceado* se a soma de seus $\lceil k/2 \rceil$ primeiros dígitos é igual à soma de seus $\lceil k/2 \rceil$ últimos dígitos. Por exemplo: $9, 101$ e 13722 são balanceados; 10 e 13723 não o são. Escreva uma função que recebe um inteiro positivo n e determina se ele é balanceado.

  Solução:

```python
from math import log10, floor

def sum_digits (m : int) -> int:
  s = 0
  while m > 0:
    s, m = s + m % 10, m // 10
  return s

def is_balanced (n: int) -> bool:
  k = 1 + floor (log10 (n)) if n > 0 else 1
  l = (k+1) // 2
  return sum_digits (n % 10**l) == sum_digits (n // 10**(k-l))
```

  ☐

- **(02)** *Inteiros Balanceados II.*
  Para um inteiro positivo m, defina $B(m)$ como a soma de todos os inteiros balanceados (veja item anterior) menores que 10^m. Por exemplo: $B(1) = 45$, $B(2) = 540$ e $B(5) = 334795890$. Escreva uma função que recebe m e determina $B(m) \mod 3^{15}$.

  Solução:
  Note que $(a + b) \bmod d = (a \bmod d + b \bmod d) \bmod d$.

```python
def balanced_summod (m: int):
  b, d = 0, 3**15
  for i in range (10**m):
    if is_balanced (i):
      b = (b + i % d) % d
  return b
```

  ☐

- **(03)** *Aproximação para* $\ln(1 + x)$.
  Dados `float`s $|x| < 1$ e $0 < \varepsilon \ll 1$, calcule uma aproximação com precisão ε para a função $\ln(1 + x)$, via série de Taylor-Maclaurin:

$$\ln(1+x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^k}{k}.$$

  Solução:

```python
def ln1p (x: float, eps: float) -> float:
  ti, tj = x, -x*x/2
  lx, k  = ti + tj, 3

  while abs (tj - ti) >= eps:
    ti, tj = tj, tj * x * (1-k)/k
    lx, k  = lx + tj, k + 1
  return tj
```

  ☐

- **(04)** *Sequências e Séries.*
  Para inteiros a, b, c, d defina a sequência infinita $G := (G_1, G_2, G_3, \ldots)$ como $G_1 = a$, $G_2 = b$ e

$$G_k = cG_{k-1} + dG_{k-2},$$

  para $k \geq 3$.

  Considere agora a série polinomial infinita

$$\mathcal{A}_G(x) := \sum_{k=1}^{\infty} = xG_1 + x^2 G_2 + x^3 G_3 + \cdots$$

  Dados a, b, c, d como acima, um inteiro $m \geq 0$ e um `float` x, escreva uma função que calcule uma aproximação para $\mathcal{A}_G(x)$ via soma dos m primeiros termos da série.

  Solução:
  Claramente, a sequência G_k é uma generalização de Fibonacci (tome $a = b = c = d = 1$). Os termos de G_k são gerados à medida que somamos os termos da série $\mathcal{A}_G(x)$.

```python
def A_G (a: int, b: int, c: int, d: int, m: int, x: float) -> float:
  if m <= 2: return (0.0, a*x, b*x*x)[m]

  gi, gj, t, s = a, b, x*x, a*x + b*x*x
  for k in range (3, m+1):
    t *= x
    g_i, gj = gj, c*gj + d*gi
    s += gj*t
  return s
```

  ☐

> Para exercícios 05 a 08:
> Seja $p \in [0, 1]$ um `float` descrevendo uma *probabilidade*. Uma moeda (com dois lados: um *cara* e o outro *coroa*) é p*-enviesada* se a probabilidade de obter-se cara após um lançamento é p, enquanto a de obter-se coroa é $1 - p$.

- **(05)** *Distribuição Binomial.*
  Dados uma probabilidade $p \in [0, 1]$ e inteiros $n \geq k \geq 0$, escreva uma função que devolve a probabilidade de obtermos k caras em n lançamentos de uma moeda p-enviesada.

  Solução:
  Seja $X \sim \mathrm{Bin}(n, p)$ uma variável aleatória que conta o número de caras em n lançamentos de uma moeda p-enviesada. Temos que

$$\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k}.$$

```python
from math import prod

def binomial (n: int, p: float, k: int) -> float:
  return ((prod (range (k+1,n+1)) // prod (range (1, k+1))) * p**k * (1-p)**(n-k)
```

  ☐

- **(06)** *Distribuição Geométrica.*
  Dados uma probabilidade $p \in [0, 1]$ e um inteiro $k \geq 0$, escreva uma função que devolve a probabilidade de obtermos a primeira cara após k lançamentos de uma moeda p-enviesada. Isto é, em $k + 1$ lançamentos, os k primeiros são coroas e o $(k + 1)$-ésimo é cara.

  Solução:
  Seja $Y \sim \mathrm{Geo}(p)$ uma variável aleatória que conta o número de lançamentos realizados até que uma moeda p-enviesada retorne cara. Temos que

$$\Pr[Y = k] = p(1-p)^k.$$

```python
def geometric (p: float, k: int) -> float:
  return p * (1-p) ** k
```

  ☐

- **(07)** *Esperança Amostral I.*
  Dados uma probabilidade $p \in [0, 1]$ e um inteiro $n \geq 0$, escreva uma função que simule n lançamentos de uma moeda p-enviesada e devolva o número de caras. Utilize as funções `seed` e `uniform` do módulo `random`.

  Solução:

```python
from random import seed, uniform

def coin_sample (p: float, n: int) -> int:
  seed()
  heads = 0
  for i in range (n):
    if uniform (0, 1) <= p: heads += 1
  return heads
```

  ☐

- **(08)** *Esperança Amostral II.*
  Dados uma probabilidade $p \in [0, 1]$ e inteiros $n, m \geq 0$, escreva uma função que execute m vezes a função do item anterior (com parâmetros p e n) e devolva a média do número de caras obtidos nos m processos (de n lançamentos cada). Nota: compare este valor com pn, o número de caras *esperado* em uma distribuição binomial.

  Solução:

```python
def sample (p: float, n: int, m: int) -> float:
  assert (m > 0 and n > 0)
  s = 0
  for i in range (m):
    s += coin_sample (p, n)
  return s/m
```

  Seja a o valor devolvido por `sample (p, n, m)`. Temos que $a \to pn$ à medida que $m \to \infty$.
  ☐

- **(09)** *Heaps.*
  Um *heap* é uma lista de inteiros $H[1\,..\,n]$ tal que $H[j//2] \geq H[j]$ para todo $2 \leq j \leq n$. Escreva uma função que recebe uma lista H e devolve `True` se H é um heap, ou `False` em caso contrário. Por exemplo: $H_1 = [5, 2, 5, 1, 1, 4, 0, 1]$ e $H_2 = [5, 4, 3, 2, 1, 0]$ são um heaps, mas $H_3 = [5, 2, 3, 3, 1]$ e $H_4 = [1, 4, 5, 2, 3, 8]$ não são heaps.

  Solução:

```python
def is_heap (H: [int]) -> bool:
  for j in range (len (H) // 2, 1, -1):
    if H[j//2] < H[j]:
      return False
  return True
```

  ☐

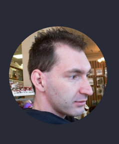
- **(10)** *Soma de dois elementos distantes.*
  Dada uma lista $L[0\,..\,n-1]$ de inteiros e um inteiro d tal que $0 \leq d \leq n - 1$, encontrar o maior número da forma $L[i] + L[j]$ com $j - i \geq d$.

  Solução:

```python
def maxsum_apart (L: [int], d: int) -> (int, int, int):
  n, mv, mi, mj = len(L), L[0] + L[0+d], 0, d

  for i in range (0, n-d+1):
    for j in range (i+d, n):
      if L[i] + L[j] > mv:
        mv, mi, mj = L[i] + L[j], i, j
  return (mv, mi, mj)
```

  ☐



Aritanan Gruber
Assistant Professor
"See, if y'all haven't the same feeling for *this*, I really don't give a damn. If you ain't feeling it, then dammit *this* ain't for you!"
(desconheço a autoria; agradeço a indicação)

cv