BCM0505-15 - Processamento da Informação

Feb 10, 2020

Laboratório L02 – 28/04

Instruções

- 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 05/05 às 19h – sob a entrada **E6.**

Exercícios

- **(01)** *Produto Palindrômico Máximo.*
  Um inteiro não negativo é um *palíndromo* se sua sequência de dígitos (decimais) for a mesma quando lida da esquerda para a direita, e da direita para a esquerda.
  O produto de dois inteiros $m, n \geq 0$ é *palindrômico* se mn for um palíndromo.
  Dado um inteiro $k \geq 1$, determine um produto palindrômico de valor máximo, em que m e n contém k dígitos (decimais).
  Exemplo: para $k = 2$, temos que $91 * 99 = 9009$ é o maior produto palindrômico formado por números de 2 dígitos.

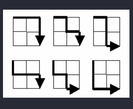
  Solução:
  Observe que, dependendo do valor de k, pode não haver um palíndromo que é produto de dois valores em $[10^{k-1}..10^k - 1]$. Neste caso, devolvemos -1.

```
def is_palindrome (m: int) -> bool:
  k, n = m, 0
  while k > 0:
    n = n * 10 + k % 10
    k = k // 10
  return m == n

def max_palindrome_prod (k: int) -> int:
  beg, end, pal = 10**(k-1), 10**k-1, -1
  for m in range (beg, end):
    for n in range (i+1, end+1):
      mn = m * n
      if is_palindrome (mn) and pal < mn:
        pal = mn
  return pal
```

  □

- **(02)** *Caminhos em Reticulados.*
  Considere uma grade 2×2. Partindo do canto superior esquerdo, com cada passo podendo ser dado uma posição à direita ou abaixo, existem 6 *rotas* (caminhos distintos) até o canto inferior direito (veja figura).



  Dado um inteiro $n \geq 0$, determine quantas rotas (como acima) existem em uma grade $n \times n$.

  Solução:
  O problema consiste em, dadas $2n$ possíveis direções, escolher n que serão à direita. Em outras palavras, queremos determinar o número de subconjuntos com n elementos à partir de um conjunto de $2n$ elementos. A resposta é: $\binom{2n}{n}$.

```
from math import prod

def couting_paths (n: int) -> int:
  return prod (range (n+1, 2*n+1)) // prod (range (1, n+1))
```

  □

- **(03)** *Segmento Máximo de Zeros.*
  Escreva uma função que recebe uma lista $A[0..n-1]$ de inteiros, calcula o comprimento m do mais longo *segmento* de zeros em A, e devolve a tripla (m, i, j), em que $m = j - i$ e todos os elementos em $A[i..j-1]$ são iguais a zero.

  Solução:

```
def longest_zero_segment (A: [int]) -> (int, int, int):
  n = len (A)
  i, j, k, l = n, n, 0, 0

  while k < n:
    while k < n and A[k] != 0: k += 1
    l = k
    while k < n and A[k] == 0: k += 1
    if k-l > j-i:
      i, j = l, k

  return (j-i, i, j)
```

  □

- **(04)** *Problema de Josephus.*
  Imagine uma roda de n pessoas. Suponha que as pessoas estão numeradas de 1 a n no sentido horário. Começando com a pessoa de número 1, percorra a roda no sentido horário e elimine cada m-ésima pessoa enquanto a roda tiver duas ou mais pessoas. Escreva uma função que calcule o número do sobrevivente.
  Exemplo: para $n = 5$ e $m = 3$, temos que

$$\langle 1,2,3,4,5 \rangle \quad \to \quad \langle 1,2,4,5 \rangle \quad \to \quad \langle 2,4,5 \rangle \quad \to \quad \langle 2,4 \rangle \quad \to \quad \langle 4 \rangle$$

  resultando no sobrevivente de número 4.

  Solução:

```
def josephus (n: int, m: int) -> int:
  C, p = [i+1 for i in range(n)], 0
  while len (C) > 1:
    p = (p + m-1) % len (C)
    del C[p]
  return C[0]
```

  □

- **(05)** *Números Distintos.*
  Dada uma lista $A[0..n-1]$ de números inteiros, determine quantos números distintos existem em A.

  Solução:

```
def distincts (A: [int]) -> int:
  if (n := len (A)) == 0: return 0
  A.sort()
  j, m = 0, 1
  for i in range (1, n):
    if A[j] != A[i]:
      j, m = i, m+1
  return m
```

  □

- **(06)** *Número de Segmentos.*
  Escreva uma função que recebe duas listas $S[0..m-1]$ e $T[0..n-1]$, com $0 \leq m \leq n$, e devolve o número de ocorrências de S como segmento em T. Lembre-se que S ocorre como um *segmento* em T se existe $0 \leq k \leq n - m$ tal que $S[0..m-1] = T[k..n+k-1]$.
  Por exemplo, se

$$S = [0,1,0] \qquad \text{e} \qquad T = [\underline{0,1,0},2,3,0,4,5,\underline{0,1,0},6,7,8,9,0,3,10,2,\underline{0,1,0,1,0}],$$

  S ocorre 4 vezes como segmento em T.

  Solução:

```
def naive_pattern_matching (S: [int], T: [int]) -> int:
  c, m, n = 0, len (S), len (T)
  for i in range(n-m+1):
    j = 0
    while j < m and S[j] == T[i+j]: j += 1
    if j == m: c += 1
  return c
```

  O algoritmo acima claramente consome tempo $\Theta(nm)$. Para algoritmos mais avançados e melhores que resolvem a mesma tarefa em tempo $O(n+m)$, procure pelos algoritmos \textsc{Knuth-Morris-Pratt} e \textsc{Boyer-Moore}. □

- **(07)** *Sub-listas.*
  Uma sub-lista de uma lista L é o que sobra depois que alguns dos elementos de L são apagados. Por exemplo, $[12,13,10,3]$ é uma sub-lista de $[11,12,13,11,10,9,7,3,3]$, mas não de $[11,12,10,11,13,9,7,3,3]$. Escreva uma função (eficiente) que decide se uma lista $A[0..m-1]$ é sub-lista de $L[0..n-1]$.

  Solução:

```
def is_sublist (A: [int], B: [int]) -> bool:
  i, j, m, n = 0, 0, len (A), len (B)
  while i < m and j < n:
    if A[i] == B[j]: i+= 1
    j += 1
  return i == m
```

  □

- **(08)** *Ordenação Indireta.*
  Escreva uma função que recebe uma lista $A[0..n-1]$ de números reais positivos e devolve uma lista $B[0..n-1]$ de índices em $\{0,1,\ldots,n-1\}$ tal que

$$A[B[0]] \leq A[B[1]] \leq A[B[2]] \leq \cdots \leq A[B[n-1]].$$

  Solução:
  Vamos utilizar o algoritmo de ordenação por seleção (*selection sort*) por simplicidade.

```
def ind_selection_sort (A : [int]) -> [int]:
  n = len (A)
  B = list (range (n))
  for i in range (n-1):
    k = i
    for j in range (i+1, n):
      if A[B[j]] < A[B[k]]: k = j
    B[i], B[k] = B[k], B[i]
  return B
```

  □

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