BCM0505-15 - Processamento da Informação

May 5, 2020

Laboratório L03 – 05/05

Instruções

- 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 12/05 às 19h – sob a entrada **E7.**

Exercícios

- **(01)** *Moda.*
  Seja $A[0\,..\,n-1]$ uma lista de inteiros. Uma *moda* de A é um elemento que mais ocorre em A. Exemplo: para $A = [0, 2, -3, 4, 2, 0, 1, 1, 5, 2, -3, -1, -3]$, tem-se que os elementos -3 e 2 são modas de A. Escreva uma função que devolva uma moda de A conjuntamente com o número de ocorrências da mesma.

  Solução:

```
def mode (A: [int]) -> (int, int):
  n = len (A); assert (n > 0)
  A.sort()

  j = k = 0
  while j < n:
    i, j = j, j+1
    while j < n and A[j] == A[i]: j += 1
    if k < j-i:
      k, m = j-i, A[i] # m é candidato à moda, ocorrendo k vezes
  return m, k
```

  O código acima determina a moda de A em tempo $O(n \log n)$. Abaixo, uma versão mais curta e assintoticamente mais lenta ($O(n^2)$).

```
def mode (A: [int]) -> (int, int):
  m = max (set (A), key = A.count)
  return (m, A.count (m))
```

  □

- **(02)** *Separação.*
  Dada uma lista (de inteiros) $A[i\,..\,j]$, com $0 \le i \le j$, defina o elemento $A[i]$ como o *pivô* de A. Escreva uma função que recebe uma lista $A[i\,..\,j]$ como acima e rearranja os elementos de A com respeito a seu pivô de forma que, ao final, tenha-se

$$A[i\,..\,k-1] \le A[k] < A[k+1\,..\,j],$$

  em que $A[k]$ é igual ao pivô de A antes do rearranjo. Observação, as sub-listas $A[i\,..\,k-1]$ e $A[k+1\,..\,j]$ não precisam estar ordenadas e não precisam conservar a ordem relativa original de seus elementos.
  Exemplo: se $A = [5, 7, 2, 1, 8, 9, 2, 3, 6, 5, 8, 0, 7]$, temos que o pivô é $A[0] = 5$. Uma separação para A com base no pivô 5 é:

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

  Neste caso, $k = 6$ e

$$[2, 0, 2, 1, 5, 3] = A[0..5] \le A[6] = 5 < A[7..12] \le [9, 6, 8, 8, 7, 7]$$

  Solução:
  Supondo que $0 \le i \le j < n$, em que $n =$ `len (A)`.
  Uma versão básica.

```
def partition (A: [int], i: int, j: int) -> int:
  c, h, k = A[i], i+1, j
  while h <= k:
    if   A[h] <= c: h += 1
    elif c <  A[k]: k -= 1
    else:
      A[h], A[k] = A[k], A[h]
      h, k = h+1, k-1
  A[i], A[k] = A[k], c
  return k
```

  Outra versão, um pouco mais sofisticada.

```
def partition (A: [int], i: int, j: int) -> int:
  p, k = A[j], j
  for l in range (i, j):
    if A[l] <= p:
      A[l], A[k] = A[k], A[l]
      k += 1
  A[j], A[k] = A[k], A[j]
  return k
```

  □

- **(03)** *Segmento de Soma Máxima.*
  Dada uma lista A de números inteiros, determinar um segmento de A cuja soma de seus elementos seja máxima. Exemplo: na lista

$$5, 2, -2, -7, \underline{3, 14, 10, -3, 9}, -6, 4, 1$$

  a soma do segmento indicado é 33; observe que nenhum outro segmento possui soma maior.

  Solução:
  Vamos determinar apenas o valor do segmento de soma máxima em tempo $O(n)$, uma solução bastante elegante!
  A alteração do código para também determinar o segmento é trivial e fica como exercício.

  Ao irmos crescendo o segmento, o segredo é observar que um segmento só deixa de ter potencial quando se torna negativo.

```
def maxsum_seg (A: [int]) -> int:
  best = curr = 0
  for x in A:
    curr = max (0, curr + x)
    best = max (best, curr)
  return best
```

  □

- **(04)** *Sub-lista Crescente Máxima.*
  Uma sub-lista de uma lista L é o que sobra depois que alguns dos elementos de L são apagados. Dada uma lista $L[0\,..\,n-1]$, determine uma sub-lista crescente de L de comprimento máximo. Exemplo: se $A = [3, 2, 4, 1, 5, 3, 6, 2, 7]$, temos que a sub-lista $[1, 2, 7]$ é crescente. Também são crescentes as sub-listas

$$[3],\ [2, 4],\ [2, 3, 6, 7],\ [2, 4, 5, 6, 7],\ [3, 4, 5, 6, 7]$$

  e várias outras. Observe que não há sub-listas crescentes de A com mais de 5 elementos. Logo, tanto $[2, 4, 5, 6, 7]$ quanto $[3, 4, 5, 6, 7]$ seriam uma resposta.

  Solução:
  Defina $C[j]$ como o número de elementos em uma sub-lista de comprimento máximo dentre todas as sub-listas crescentes cujo primeiro elemento seja $L[j]$. Claramente, $C[j] \le n - j + 2$ para todo $0 \le j \le n-1$. Considerando a subestrutura ótima de uma solução, temos que

$$C[j] = \begin{cases} 0 & \text{se } j = n, \\ \max\{1 + C[k] : j < k \le n \text{ e } C[j] \le C[k]\} & \text{para } 0 \le j < n, \end{cases}$$

  em que $C[n] = 0$ é uma sentinela significando $L[n] = \infty$.

  Agora, não é difícil perceber que a relação de recorrência acima fornece, imediatamente, um algoritmo com consumo de tempo $O(n^2)$ para o problema em questão.

```
def max_increasing_sublist (L: [int]) -> (int, int, [int]):
  n = len (L):
  C = [0 for j in range (n+1)]
  s, i = 0, n

  for j in range (n-1, -1, -1):
    for k in range (n, j, -1):
      if C[j] < 1 + C[k] and L[j] <= L[k]:
        C[j] = 1 + C[k]
        if s < C[j]:
          s, i = C[j], j
  return (s, i, C)
```

  A função acima devolve s, o comprimento da sub-lista crescente (não estrita) mais longa em L, o índice $0 \le i < n$ do primeiro elemento desta sub-lista e um vetor C que contém a estrutura das sub-listas crescentes mais longas (cadeias) iniciadas em cada elemento $L[j]$, para $0 \le j < n$.

  A função abaixo utiliza C para imprimir a sub-lista de comprimento s que começa em i.

```
def print_sublist (L: [int], C: [int], s: int, i: int):
  for j in range (i, len (L)):
    if C[j] == s:
      print (L[j])
      s -= 1
  return
```

  □

- **(05)** *Enumeração de Sub-listas.*
  Escreva uma função que recebe um inteiro $n \ge 1$ e imprime, em ordem lexicográfica, todas as sub-listas não vazias de $\{1, 2, \ldots, n\}$.
  Exemplo: se $n = 3$, deve ser impresso

```
1
1 2
1 2 3
1 3
2
2 3
3
```

  Solução:

```
def sublists (n: int) -> [int]:
  assert (n >= 1)
  k, s = 0, [i for i in range (0, n+1)]

  while True:
    if s[k] < n:
      s[k+1] = s[k] + 1; k += 1
    else:
      s[k-1] += 1; k -= 1

    if k == 0: break
    yield s[:k+1]
```

  Para teste:

```
for q in sublists (3):
  print (q)
```

  Nota: para transladar as sub-listas à esquerda e eliminar o 0, substitua o argumento do `yield` por `s[1:k+1]`.
  □

- **(06)** *Enumeração de Tuplas.*
  Escreva uma função que recebe inteiros $n, k \ge 1$ e imprime, em ordem lexicográfica, todas as tuplas de k elementos de $\{1, 2, \ldots, n\}$.
  Exemplo: se $n = 4$ e $k = 2$, deve ser impresso

```
1 2
1 3
1 4
2 3
2 4
3 4
```

  Solução:

```
def combine (n: int, m: int) -> [int]:
  assert (n >= 1 and m >= 1)
  i, s = m, [i for i in range (m+1)]

  while i > 0:
    i = m
    yield s[:m+1]

    while i > 0 and s[i] == n-m+i: i -= 1
    if i == 0: break

    x = s[i] + 1
    while i <= m:
      s[i] = x
      i, x = i+1, x+1
```

  Para teste:

```
for q in combine (4, 2):
  print (q)
```

  Nota: para transladar os subconjuntos à esquerda e eliminar o 0, substitua o argumento do `yield` por `s[1:m+1]`.
  □

- **(07)** *Matrizes Simétricas.*
  Uma matriz em Python pode ser interpretada como uma lista de listas. Assim, a matriz

$$A = \begin{pmatrix} -1 & 0 & 0 & 3 \\ 2 & 5 & 8 & 0 \\ 3 & 0 & 1 & -4 \end{pmatrix}$$

  é equivalente ao código

```
  A = [
        [-1, 0, 0, 3],
        [ 2, 5, 8, 0],
        [ 3, 0, 1,-4]
      ]
```

  ou, de forma mais sucinta, a

```
A = [[-1,0,0,3], [2,5,8,0], [3,0,1,-4]]
```

  O elemento A_{ij}, com i e j começando de zero, é acessado via `A[i][j]`. Exemplo: $A_{12} = 8 =$ `A[1][2]`.

  Uma matriz é *quadrada* se os números de linhas e colunas são iguais. Uma matriz quadrada A é *simétrica* se $A_{ij} = A_{ji}$ – isto é, se A for igual à sua *transposta*, em que a troca das linhas pelas colunas resulta na mesma matriz. Escreva uma função que recebe uma matriz quadrada A e determina se A é simétrica.

  Solução:

```
def is_symmetric (A: [[float]]) -> bool:
  for i in range (len (A)):
    for j in range (i):
      if A[i][j] != A[j][i]:
        return False
  return True
```

  □

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