Soluções para a Lista 1
Processamento da Informação – 2020.1

Universidade Federal do ABC

Aritanan Gruber

aritanan.gruber@ufabc.edu.br

http://professor.ufabc.edu.br/∼aritanan.gruber

Entregue os exercícios sinalizados por [⋆] via Tidia-4 até às 10/03/2018 às 09h00.

1. *Anos bissextos.* No calendário Gregoriano, um ano a é *bissexto* — o mês de Fevereiro tem 29 dias, em vez de 28 — se a é divisível por 4 mas não por 100, ou se a é divisível por 400. Escreva uma função que recebe um inteiro a e devolve verdadeiro se a corresponde a um ano bissexto, ou falso em caso contrário.

   Solução:

```python
def is_leapyear (a: int) -> bool:
  return a % 4 == 0 and a % 100 != 0 or a % 400 == 0
```

2. [⋆] *Dia da semana.* Escreva uma função que recebe três inteiros d, m, e a, representando dia ($1 \leq d \leq 31$), mês ($1 \leq m \leq 12$) e ano ($a \geq 1$) e devolve o dia da semana em que a data informada ocorreu ou ocorrerá. Para tal, utilize as fórmulas apropriadas para o calendário Gregoriano:

$$
\begin{aligned}
a_0 &= a - (14 - m)//12 \\
x &= a_0 + a_0//4 - a_0//100 + a_0//400 \\
m_0 &= m + 12 * ((14 - m)//12) - 2 \\
d_0 &= (d + x + (31 * m_0)//12) \ \% \ 7
\end{aligned}
$$

   Considere que d_0 igual a 0 representa "Domingo", igual a 1, "Segunda", e assim sucessivamente.

   Solução:

   As fórmulas já foram dadas na ordem em que devem ser avaliadas.

```python
# recebe uma data (dia, mês, ano) e devolve seu dia da semana (entre 0 e 6)
def weekday (d: int, m: int, a: int) -> int:
  a0 = a - (14 - m) // 12
  x  = a0 + a0//4 - a0//100 + a0//400
  m0 = m + 12 * ((14 - m) // 12) - 2
  return (d + x + (31 * m0) // 12) % 7
```

   A resposta acima já era suficiente, mas vamos aproveitar a oportunidade e exibir uma função envólucro que verifica a consistência da data fornecida e imprime o dia da semana correspondente por extenso.

```python
def dia_da_semana (d: int, m: int, a: int) -> str:
  dias_mes = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
  if not ((a >= 1) and (1 <= m <= 12) and (1 <= d <= dias_mes[m])):
    return "data inválida"
  else:
    dias = ("Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sáb")
    return dias[weekday (d, m, a)]
```

3. *Identidade de Bézout.* Escreva uma função que recebe dois inteiros positivos m e n, e devolve três inteiros d, a e b tais que d é o máximo divisor comum a m e n e $am + bn = d$.

   Solução: Apresentamos uma dedução completa.

   Suponha, sem perda de generalidade, que $m \geq n$ e defina $r_0 := m$ e $r_1 := n$. Uma simples indução em $k \geq 1$ mostra que o k-ésimo passo do algoritmo de Euclides garante que

$$r_{k+1} = r_{k-1} - r_k q_k \qquad \text{e} \qquad 0 \leq r_{k+1} < |r_k|. \tag{1}$$

   Além disto, $d := r_k = \mathsf{mdc}(m, n)$ quando $r_{k+1} = 0$.

   Por hipótese, temos que

$$ma_k + nb_k = r_k \qquad \text{para} \qquad k \in \{0, 1\}, \tag{2}$$

   implicando em $a_0 = 1$, $b_0 = 0$, $a_1 = 0$ e $b_1 = 1$. Utilizamos estes valores como caso base para uma nova indução em k. Suponha então que (2) é válida para $k > 1$. Por (1), temos que

$$\begin{aligned} r_{k+1} &= r_{k-1} - r_k q_k \\ &= (ma_{k-1} + nb_{k-1}) - (ma_k + nb_k)q_k \\ &= m(a_{k-1} - a_k q_k) + n(b_{k-1} - b_k q_k) \\ &= ma_{k+1} + nb_{k+1}, \end{aligned}$$

   com

$$a_{k+1} = a_{k-1} - a_k q_k, \tag{3}$$
$$b_{k+1} = b_{k-1} - b_k q_k. \tag{4}$$

   Modificando o algoritmo de Euclides para que as recorrências (1), (3) e (4) sejam computadas, determinamos a e b (*coeficientes de Bézout*) como desejados: $a = a_k$ e $b = b_k$.

```python
def extended_euclides (m: int, n: int) -> (int, int, int):
  ri, rj = max(m,n), min(m,n)
  ai, aj = 1, 0
  bi, bj = 0, 1
  while True:
    qj, rk = ri//rj, ri%rj
    ai, aj = aj, ai - aj*qj
    bi, bj = bj, bi - bj*qj
    if rk == 0: return rj, aj, bj
    ri, rj = rj, rk
```

4. *Aproximação de* arcsin. Dados $|x| \leq 1$ e $0 < \varepsilon \ll 1$ reais, calcular uma aproximação para $\arcsin x$ (o arco cujo seno é x) através da série infinita de Taylor-Maclaurin ao redor de 0:

$$\arcsin x = x - \frac{x^3}{6} + \frac{3x^5}{40} + \ldots + \frac{(2k)!}{4^k (k!)^2 (2k+1)} x^{2k+1} + \ldots$$

   com precisão ε, isto é, até que a diferença absoluta de dois termos consecutivos seja menor que ε:

$$\left| \frac{(2k)!}{4^k (k!)^2 (2k+1)} x^{2k+1} - \frac{(2k-2)!}{4^{k-1} ((k-1)!)^2 (2k-1)} x^{2k-1} \right| < \varepsilon.$$

   Solução:

A resposta corresponde à soma de termos até que a diferença absoluta de dois termos adjacentes seja menor que a precisão estabelecida por ε; como os valores somados são cada vez menores, o critério de parada é alcançado mais cedo ou mais tarde.

O termo genérico da série de arcsin ao redor de 0 é dado por:

$$T(k) := \frac{(2k)!\,x^{2k+1}}{4^k(k!)^2(2k+1)}.$$

Dividindo $T(k+1)$ por $T(k)$, temos

$$R(k) := \frac{T(k)}{T(k-1)} = \frac{(2k)!\,x^{2k+1}}{4^k(k!)^2(2k+1)} \cdot \frac{4^{k-1}((k-1)!)^2(2(k-1)+1)}{(2(k-1))!\,x^{2(k-1)+1}} = \frac{(2k)(2k-1)^2}{4k^2(2k+1)}\,x^2,$$

o que resulta em $T(k) = T(k-1)R(k)$.

Um possível código em Python:

```
def apx_arcsin (x: float, eps: float) -> float:
  def next_term (t: float, k: int, x: float) -> float:
    return t*(2*k)*(2*k-1)*(2*k-1)*x*x/(4*k*k*(2*k+1))

  k, ti, tj = 1, x, next_term (ti, 1, x)
  arcsin_x  = ti + tj
  while (abs(tj-ti) > eps):
    k += 1
    ti, tj = tj, next_term (ti, k, x)
    arcsin_x += tj
  return arcsin_x
```

Exemplo de uso:

```
print ("Aproximação de arcsin(sqrt(3)/2) com 6 casas decimais: ")
print (apx_arcsin (math.sqrt(3)/2, 1e-6))
```

5. [⋆] *Binômios de Newton.* Escreva uma função que recebe um inteiro $n \geq 0$ e imprime a expansão do binômio $(x+y)^n$. Por exemplo, para $n = 3$, $(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$ e a saída esperada é: `x**3 + 3*x**2*y + 3*x*y**2 + y**3`. Para tal, utilize a fórmula de expansão do binômio de Newton:

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i}y^i = x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n-1}xy^{n-1} + y^n.$$

Solução:

Inicialmente, uma função para calcular os coeficientes binomiais que ocorrem na expansão do binômio.

```
# recebe n >= m >= 0 e devolve \binom{n}{m}
def binomial_coef (n: int, m: int) -> int:
  a, b = 1, 1
  for i in range (n-m+1,n+1): a *= i
  for j in range (2,m+1):     b *= j
```

O código abaixo está correto, mas não segue as especificações de formatação desejada para a saída.

```
def preliminar_binomial (n: int):
  s = ""
  for i in range (0,n+1):
    c = binomial_coef (n, i)
    s += " + " + str(c) + "*x**" + str(n-i) + "*y**" + str(i)
  print (s)
```

De fato, para $n = 3$, esta versão imprime:

`+ 1*x**3*y**0 + 3*x**2*y**1 + 3*x**1*y**2 + 1*x**0*y**3`.

Com um pouco mais de cuidado – e à custa de alguns `if`s – obtemos o resultado solicitado.

```
1  def expand_binomial (n: int):
2    s = ""
3    for i in range (0,n+1):
4      c = binomial_coef (n, i)
5      if i > 0:              s += " + "
6      if c > 1:              s += str(c) + "*"
7      if n-i > 0:            s += "x"
8      if n-i > 1:            s += "**" + str(n-i)
9      if n-i > 0 and i > 0:  s += "*"
10     if i > 0:              s += "y"
11     if i > 1:              s += "**" + str(i)
```

Nota: a função `str` converte o argumento recebido em uma *string*, que pode ser concatenada ao final de outra string pelo operador `+`.

6. *Triângulos.* Escreva uma função que recebe três valores a, b e c e determina se eles são lados de um triângulo. Em caso afirmativo, classifique o triângulo que eles formam como: 1 - escaleno, 2 - isósceles, 3 - equilátero. Em caso negativo, devolva 0.

   Solução:

```
1  def is_triangle (a: int, b: int, c: int) -> int:
2    if not (a <= b+c and b <= a+c and c <= a+b): return 0
3    if a == b == c: return 3
4    elif a == b or a == c or b == c: return 2
5    else: return 1
```

7. [⋆] *Busca sequencial.* Considere a variante abaixo da busca sequencial, que recebe uma lista $A[0..n-1]$ de inteiros e um inteiro x e promete devolver um índice j tal que $A[j] =$ se $x \in A$, ou $j = n$ em caso contrário. Esta variante faz o prometido? Como? Qual o papel das linhas 2 e 5?

```
1  def busca_sequencial (A: [int], x: int) -> int:
2    A.append (x)
3    j = 0
4    while A[j] != x: j += 1
5    A.pop()
6    return j
```

   Solução: O código acima implementa a busca sequencial em uma lista com sentinela. O elemento procurado é anexado ao final da lista A (i.é, na posição n) pela instrução `A.append (x)` da linha 1. Com isso, o laço `while` da linha 2 certamente irá determinar um índice j tal que $A[j] = x$: ao final do laço, teremos $j < n$ caso x ocorra em A, ou $j == n$ em caso contrário. Finalmente, a linha 5 remove o sentinela x anexado ao final de A, deixando A nas condição prévia à chamada da função.

8. [⋆] *Embaralhamento?* Considere a seguinte antítese do problema de ordenação: fazer um embaralhamento aleatório dos elementos de uma lista $A[0..n-1]$, ou seja, rearranjar os elementos da lista de modo que todas as permutações sejam igualmente prováveis. Discuta e critique a seguinte solução (cuja variante foi usada na distribuição europeia do Windows 7 da Microsoft).

```
1  import random
2
3  def random_insert (A: [int]):
4    for j in range(1, len (A)):
5      x = A[j]
6      for i in range (j-1,-1,-1):
7        if random.random() > 0.5:
8          A[i+1] = A[i]
9        else:
10         break
11     A[i+1] = x
```

Solução:

Dada uma lista $A[0..n-1]$, o problema do *embaralhamento* de A consiste em devolver uma permutação de A de acordo com uma medida uniforme de probabilidade. Em outras palavras, qualquer uma das $n!$ permutações de A existentes deve ter chance $1/n!$ de ser fornecida como resposta.

Uma análise do código acima revela que ele não resolve tal problema: apesar de modificar as posições relativas entre alguns elementos em A, ele não garante uniformidade na permutação devolvida. Isso decorre do fato do elemento em $A[0]$ permanecer inalterado. Vejamos o porquê.

A lista A é modificada somente pelas instruções `A[i+1] = A[i]` na linha 9 e `A[i+1] = x` na linha 12. Em ambos os casos, o menor valor que a variável i atinge é 0, pois i é iterada em $[j-1,\ldots,0]$. Logo, a menor posição alterada é $A[1]$. Isso garante que a permutação devolvida é sorteada de um conjunto com $(n-1)!$ elementos em vez de $n!$, o que se traduz a qualquer permutação de A cujo primeiro elemento é diferente do valor originalmente contido em $A[0]$ tem chance 0 de ser devolvida.

A resposta acima é suficiente para concluir o exercício, mas vamos um pouco mais longe nesta solução.

Esse erro é de simples conserto: acrescente uma cláusula `else: i = -1` ao laço `for i` entre as linhas 11 e 12. Independente disso e, de qualquer forma, uma pergunta paira no ar: é o caso de que todas as $(n-1)!$ permutações com $A[0]$ como primeiro elemento têm a mesma chance $1/(n-1)!$ de ser devolvida?

A resposta é negativa: algumas permutações são muito mais prováveis que outras. Fornecemos um argumento simples e ligeiramente informal para essa afirmação.

Para que uma permutação $\pi(A)$ de A seja gerada uniformemente, é preciso que quaisquer dois elementos $A[r]$ e $A[s]$, com $r < s$, tenham a mesma chance $1/2$ de ocorrer em $\pi(A)$ com a ordem relativa entre eles preservada ou revertida. Isso garante que todo elemento de A tem a mesma chance $1/n$ de ocorrer em qualquer uma das n posições de $\pi(A)$.

A transposição dos elementos em A é feita pelo código acima como segue. O elemento $A[j]$ é inserido numa posição $k \in \{1,2,\ldots,j\}$, determinada aleatoriamente, com os elementos em $A[k..j-1]$ sendo escorregados para $A[k+1..j]$ antes da inserção. O problema é que o processo aleatório que escolhe o valor de k não é uniforme.

A função `random.random()` devolve um `float` uniformemente distribuído em $[0,1)$. A instrução `if random.random() > 0.5:` funciona então como o lançamento de uma moeda honesta: caso o resultado seja *cara*, a instrução `A[i+1] = A[i]` é executada e o laço prossegue; caso seja *coroa*, o laço é interrompido.

Esse mecanismo produz uma escolha para k que segue uma *distribuição binomial* com parâmetros j e $p = 1/2$, cuja *esperança* é $jp = j/2$. Ou seja, em média, $A[j]$ será inserida na posição $\lfloor j/2 \rfloor$. A *variância* do processo é $jp(1-p) = j/4$. A chance de dois elementos com distância j terem suas ordens relativas revertidas é

$$\Pr\left[|k - j/2| \geq j/2\right] \leq \frac{j/4}{(j/2)^2} = \frac{1}{j} < \frac{1}{2}$$

para $j > 2$. Em outras palavras, a chance de dois elementos terem sua ordem relativa revertida pelo código acima fica cada vez menor com o aumento da distância entre eles.

9. [⋆] *Busca binária.* Considere o algoritmo abaixo que recebe um inteiro x e uma lista crescente $v[0..n-1]$ e devolve um índice d em $\{0, 1, \ldots n\}$ tal que $v[d-1] < x \leq v[d]$.

```
def busca_binaria (x: int, v: [int]) -> int:
  e, d = -1, len(v)
  while e < d-1:
    m = (e + d)//2
    if v[m] < x:
      e = m
    else:
      d = m
  return d
```

Responda as seguintes perguntas sobre a função `busca_binaria`. O que acontece se

(a) `while e < d-1:` for trocado por `while e < d:` ou por `while e <= d-1:`?

Solução: As modificações `while e < d:` e `while e <= d-1:` são equivalentes, resultando no mesmo comportamento. A posição d não pertence à lista v. Se o elemento x for maior que todos os demais elementos em v, ocorrerá um acesso indevido à posição $v[d]$ da memória e o programa será abortado com erro.

(b) trocarmos `(e + d)//2` por `(e + d - 1)//2` ou por `(e + d + 1)//2` ou por `(d - e)//2`?

Solução: O valor `m = (d - e)//2` pode ser menor que e, fazendo com que o elemento $v[m]$ acessado esteja à esquerda da sub-lista $v[e..d-1]$. O algoritmo então se concentrará nas posições iniciais de v e será incapaz de determinar se x ocorre em posições próximas ao final de v; uma resposta incorreta será fornecida neste caso.

Para `(e + d - 1)//2` e `(e + d + 1)//2`, a questão é de arredondamento: o subintervalo da esquerda ou da direita pode ficar ligeiramente menor que a sua contra-parte, respectivamente.

(c) `if (v[m] < x)` for trocado por `if (v[m] <= x)`?

Solução: Seria devolvido $j \in [0, 1, \ldots, n]$ tal que $v[j-1] \leq x < v[j]$.

(d) `e = m` for trocado por `e = m+1` ou por `e = m-1`?

Solução: O índice j devolvido estaria uma posição à direita do último x ou uma posição à esquerda do primeiro x em v, respectivamente.

(e) `d = m` for trocado por `d = m+1` ou por `d = m-1`?

Solução: O índice j devolvido estaria uma posição à esquerda do último x ou uma posição à direita do primeiro x em v, respectivamente.

10. *Inversões.* Seja $A[0..n-1]$ uma lista de $n \geq 0$ números distintos. Se $0 \leq i < j < n$ e $A[i] > A[j]$, então o par (i, j) é chamado de uma *inversão* em A. Por exemplo, se $A = [2, 3, 8, 6, 1]$, os cinco pares $[(0, 4),\ (1, 4),\ (2, 3),\ (2, 4),\ (3, 4)]$ são inversões em A. Escreva uma função que recebe A e devolve uma lista contendo as inversões existentes em A.

Solução:

Um algoritmo consistem em: (i) gere todos os pares (i, j) tais que $0 \leq i < j < n$; (ii) para cada par gerado, verifique se $A[i] > A[j]$; (iii) em caso afirmativo, o par (i, j) é uma inversão; (iv) em caso negativo, ele pode ser desconsiderado.

Uma codificação possível:

```
# recebe uma lista A[0..n-1] e devolve uma lista contendo as inversões em A.
def invertions (A: [int]) -> [int]:
  Inv, n = [], len(A)
  for i in range (0,n-1):
    for j in range (i+1,n):
      if A[i] > A[j]:
        # o par (i,j), uma inversão, é incluido no final da lista Inv.
        Inv.append ((i,j))
  return Inv
```

Nota 1: A função `Inv.append ((i,j))` inclui o argumento (i, j) ao final da lista `Inv`.

Nota 2: O código acima determina todas as inversões em tempo $\Theta(n^2)$. Utilizando as ideias introduzidas na ordenação por intercalação (*merge-sort*), é possível realizar a tarefa em $\Theta(n \log n)$.

Suponha que a função `merge (A: [int], p: int, q: int, r: int) -> int` foi estendida de forma a receber uma lista A e inteiros $p < q < r$ tal que $A[p..q-1]$ e $A[q..r-1]$ estão ordenadas, rearranjar os elementos de $A[p..r-1]$ em ordem crescente e devolver o número de inversões encontrados neste processo. `merge` ainda consome tempo $\Theta(r-p)$.

Podemos modificar a função `merge_sort` de forma a, além de ordenar A, devolver o número de inversões encontradas durante a ordenação.

```
def merge_sort (A: [int]) -> int:
  def ms (A: [int], p: int, r: int) -> int:
    if p >= r-1: return 0
    q  = (p + r)//2
    return ms (A, p, q) + ms (A, q+1, r)
  return ms (A, 0, len (A))
```

Exercício: modifique o código de `merge` visto em sala para satisfazer as hipóteses acima.