TÉCNICO LISBOA

Fundamentos da Programação
Ano lectivo 2020-21
Primeiro Exame
3 de Fevereiro de 2021

# Pergunta 1 (val. 2)

Considere a linguagem cujas frases começam pelo símbolo 'a', o qual é seguido por um número par de ocorrências de um dos símbolos 'b', 'c', 'd', após o que terminam com o símbolo 'e'. Por exemplo 'abcddcde' e 'abcdbcdbcdbcde' são frases da linguagem; 'ae' e 'ababe' não o são.
**a)** Escreva uma gramática em notação BNF para a linguagem apresentada.

```
<frase> ::= a <par> e
<par>   ::= <char> <char> | <char> <char> <par>
<char>  ::= b | c | d
```

**b)** Escreva o predicado reconhece que recebe como argumento uma cadeia de caracteres e devolve verdadeiro apenas se a cadeia de caracteres pertence à linguagem. O predicado gera um erro se o seu argumento não for uma cadeia de

```
def reconhece(seq) :
        return type(seq) == str and len(seq) % 2 == 0 \
                and len(seq) > 3 and seq[0] == 'a' and seq[-1] == 'e' \
                and all([ ch in 'bcd' for ch in seq[1:-1] ])
```

# Pergunta 2 (val. 1.5)

Escreva uma função que recebe um número inteiro positivo e que calcula uma codificação para esse número do seguinte modo:

- a ordem dos algarismos do número é invertida;
- cada algarismo ímpar é substituído pelo ímpar seguinte, entendendo-se que o ímpar seguinte a 9 é 1;
- cada algarismo par é substituído pelo par anterior, entendendo-se que o par anterior a 0 é 8.

Caso o argumento não esteja correto, deverá gerar um erro com o texto 'Argumento invalido'. Por exemplo:

```
>>> codifica(1234567890)
8169472503
```

```
def codifica(num):
    if not (type(num)==int and num>=0):
        raise ValueError('Argumento invalido')
    novo_num = 0
    while num > 0:
```

```
        dig = num % 10
        if dig % 2 == 0: # par
            dig = (dig - 2) % 10
        else: # impar
            dig = (dig + 2) % 10
        novo_num = novo_num * 10 + dig
        num = num // 10
    return novo_num
```

# Pergunta 3 (val. 2)

Suponha que existe a função de um argumento n_esimo_primo, que recebe um número natural e devolve o número primo cuja posição corresponde ao seu argumento. Por exemplo n_esimo_primo(1) devolve 2, n_esimo_primo(2) devolve 3, n_esimo_primo(3) devolve 5, e assim sucessivamente.

**(a)** Escreva a função codifica_palavra que recebe como argumento uma palavra e devolve a codificação dessa palavra usando os números de Gödel. Uma palavra com n letras (letra_1 letra_2 ... letra_n) codificada usando os números de Gödel corresponde ao inteiro 2**ord(letra_1)*3**ord(letra_2)* ... *n_esimo_primo(n)**ord(letra_n). Por exemplo a palavra 'dia' é codificada como 2**ord('d')*3**ord('i')*5**ord('a'). Não é necessário validar o argumento da sua função.

```
def codifica_palavra(palavra) :
        res = 1
        seq = 1
        for ch in palavra:
                res *= n_esimo_primo(seq) ** ord(ch)
                seq += 1
        return res
```

**(b)** Escreva a função descodifica_palavra que recebe como argumento um natural correspondente a uma palavra codificada usando os números de Gödel e devolve a palavra original. Não é necessário validar os dados de entrada

```
def descodifica_palavra(natural) :
        res = ''
        seq = 1
        while natural > 1 :
                primo = n_esimo_primo(seq)
                ch = 0
                while natural % primo == 0 :
                        ch += 1
                        natural //= primo
                res += chr(ch)
                seq += 1
        return res
```

# Pergunta 4 (val. 1)

Escreva a função duplica_elementos_pares que recebe uma lista de inteiros e devolve a lista obtida da lista original em que todos os elementos pares são duplicados. Não é necessário validar os dados de entrada. Por exemplo:

```
>>> duplica_elementos_pares([2, 3, 4, 5, 6, 7])
[2, 2, 3, 4, 4, 5, 6, 6, 7]
```

```
def duplica_elementos_pares(lst):
        nova_lst = []
        for elem in lst:
                nova_lst += [elem]
                if elem % 2 == 0:
                nova_lst += [elem]
        return nova_lst
```

# Pergunta 5 (val. 1)

Escreva a função apenas_digitos_pares que recebe um número inteiro não negativo n, e devolve um inteiro composto apenas pelos dígitos pares de n. Se n não tiver dígitos pares, a função deve devolver zero. Não pode usar cadeias de caracteres. Não é necessário validar os dados de entrada. Por exemplo:

```
>>> apenas_digitos_pares(664426383905433)
664426804
```

```
def apenas_digitos_pares(num):
    novo_num, b = 0, 0
    while num > 0:
        dig = num % 10
        if dig % 2 == 0:
            novo_num += dig * 10**b
            b += 1
        num = num // 10
    return novo_num
```

# Pergunta 6 (val. 1)

Suponha que existe predicado de um argumento eh_primo, que recebe um número natural e devolve verdadeiro apenas se o seu argumento é um número primo. Usando um ou mais dos funcionais sobre listas (filtra, transforma, acumula), escreva a função conta_primos, que recebe uma lista de inteiros, e devolve o número de elementos da lista que são primos. A sua função deve conter apenas uma instrução, a instrução return. Não é necessário validar os dados de entrada. Por exemplo:

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

```
def conta_primos(lst):
    return acumula(lambda x,y: x+y, \
        transforma(lambda x:1, filtra(eh_primo,lst)))
```

# Pergunta 7 (val. 3)

Escreva a função soma_digitos_impares que recebe um número inteiro positivo n, e devolve a soma de dígitos pares de n. As suas funções não podem usar cadeias de caracteres. Não necessita de verificar a validade dos argumentos. Por exemplo:

```
>>> soma_digitos_impares(92)
9
>>> soma_digitos_impares(18126)
2
```

**a)** Usando recursão com operações adiadas (não pode utilizar a atribuição nem os ciclos while e for).

```
def soma_digitos_impares_oa(num):
    if num==0:
        return 0
    elif num % 2 != 0:
        return num%10 + soma_digitos_impares_oa(num//10)
    else:
        return soma_digitos_impares_oa(num//10)
```

**b)** Usando recursão de cauda (não pode utilizar a atribuição nem os ciclos while e for).

```
def soma_digitos_impares_rc(num):
    def aux(num, res):
        if num == 0:
            return res
        elif num % 2 != 0:
            return aux(num//10, res + num%10)
        else:
            return aux(num//10, res)
    return aux(num, 0)
```

**c)** Usando um processo iterativo.

```
def soma_digitos_impares_it(num):
    soma = 0
    while num > 0:
        if num % 2 != 0:
```

```
            soma += num % 10
        num = num // 10
    return soma
```

# Pergunta 8 (val. 2)

Escreva a função soma_dicionarios que recebe dois dicionários, cujos valores associados às chaves correspondem a listas, e devolve o dicionário que contém todas as chaves contidas em pelo menos um dos dicionários e o valor associado a cada chave corresponde à lista obtida pela união (no sentido de conjuntos) das listas correspondendo às chaves existentes nos dicionários. Por exemplo:

```
>>> d1 = {'a' : [1, 2], 'b' : [3, 4]}
>>> d2 = {'b' : [4, 5], 'c' : [6, 7]}
>>> soma_dicionarios (d1, d2)
{'a': [1, 2], 'b': [3, 4, 5], 'c': [6, 7]}
```

```
def soma_dicionarios(d1, d2):
    nd = {k:d1[k][:] for k in d1}

    for k in d2:
        if k in nd:
            for e in d2[k]:
                if e not in nd[k]:
                    nd[k]= nd[k] + [e]
        else:
            nd[k] = d2[k][:]

    return nd
```

# Pergunta 9 (val. 3)

Uma árvore binária é um tipo de dados que corresponde a uma estrutura hierárquica. Uma árvore binária ou é vazia ou é constituída por uma raiz que domina duas árvores binárias, a árvore esquerda e a árvore direita.
O tipo árvore binária tem as seguintes operações básicas, as quais se referem ao tipo elemento que corresponde ao tipo dos elementos da raiz:
*Construtores:*

- nova_arv(): tem como valor uma árvore vazia.
- cria_arv(r, ae, ad), em que r é do tipo elemento e ae e ad são árvores binárias. Tem como valor a árvore binária com raiz r, com árvore esquerda ae e com árvore direita ad.

*Seletores:*

- raiz(a), recebe uma árvore binária, a, e tem como valor a sua raiz. Se a árvore for vazia, o valor desta função é indefinido.
- arv_esq(a), recebe uma árvore binária, a, e tem como valor a sua árvore esquerda. Se a árvore for vazia, o valor desta operação é indefinido.
- arv_dir(a), recebe uma árvore binária, a, e tem como valor a sua árvore direita. Se a árvore for vazia, o valor desta operação é indefinido.

*Reconhecedores:*

- eh_arv(arg), recebe como argumento um elemento de um tipo qualquer e decide se este pertence ou não ao tipo árvore binária.
- eh_arv_vazia(a), recebe uma árvore binária, a, e tem o valor verdadeiro se a é uma árvore vazia e tem o valor falso, em caso contrário.

Não consideramos testes nas operações básicas.
**a)** Escolha uma representação para árvores binárias
$\mathcal{R}[arvore\_bin] = tuple(raiz, arv\_esq, arv\_dir)$
**b)** Escreva as operações básicas em termos da sua representação

```
def nova_arv():
        return ()

def cria_arv(r, ae, ad):
        return r, ae, ad

def raiz(a):
        return a[0] if a else None

def arv_esq(a):
        return a[1] if a else None

def arv_dir(a):
        return a[2] if a else None

def eh_arv(arg):
        return type(arg) == tuple and (len(arg) == 0 or \
                len(arg) == 3 and eh_arv(arg[1]) and eh_arv(arg[2]))

def eh_arv_vazia(a):
        return len(a) == 0
```

**c)** Escreva o predicado arv_iguais que recebe duas árvores binárias e devolve verdadeiro apenas se as árvores recebidas são iguais. Duas árvores são iguais, se forem ambas vazias ou se tiverem a mesma raiz e as árvores esquerda e direita forem recursivamente iguais.

```
def arv_iguais(a1, a2):
        if not eh_arv(a1) or not eh_arv(a2):
                raise ValueError
        if eh_arv_vazia(a1) and eh_arv_vazia(a2):
                return True
        if eh_arv_vazia(a1) or eh_arv_vazia(a2):
                return False
        if raiz(a1) != raiz(a2):
                return False
        return arv_iguais(arv_esq(a1), arv_esq(a2)) and \
                arv_iguais(arv_dir(a1), arv_dir(a2))
```

# Pergunta 10 (val. 2)

Defina a classe piscina que simula o comportamento de uma piscina pública. A piscina tem uma lotação máxima e uma tarifa, de acordo com a seguinte regra: €2 fixo inicial mais €0.5 por cada hora ou fração, com um custo máximo de €5. Por exemplo, se uma pessoa estiver 3 horas e 20 minutos na piscina irá pagar € 4.00 (€2 iniciais, mais € 1,50 pelas primeiras 3 horas e 0,5 pela fração seguinte). Assuma que sempre que se acede à piscina o BI da pessoa que entra é associado à hora em que entrou (horas e minutos). Não se considera a data de entrada pois todos os banhistas são expulso à meia noite. Sempre que se sai da piscina é indicada a hora de saída e calcula-se a quantia a pagar. As instâncias desta classe são criadas indicando a lotação máxima da piscina. Os métodos disponíveis são os seguintes:

- entra(bi, h, m) regista que a pessoa identificada pelo seu bi como uma cadeia de carateres entrou na piscina às h horas e m minutos. Este método verifica a legalidade das horas e minutos de entrada e devolve o número de pessoas na piscina. Se a piscina estiver sem lugares livres gera uma mensagem. Se uma pessoa com o mesmo BI já estiver na piscina, gera uma mensagem e a pessoa não entra.
- sai(bi, h, m) regista que o pessoa com BI bi saiu da piscina às h horas e m minutos. Este método verifica a legalidade das horas e minutos de saída, comparando-a com as horas e minutos de entrada. Devolve o valor a pagar. Se a piscina estiver vazia gera uma mensagem. Se o pessoa com o BI não estiver na piscina, gera uma mensagem.
- ocupacao() devolve o número de pessoas dentro da pisina.

Por exemplo:

```
>>> p = piscina(3)
>>> p.entra('5555555', 5, 12)
1
>>> p.entra('5555555', 12, 30)
'entra: a pessoa está na piscina'
>>> p.sai('5555555', 5, 10)
```

```
'sai: horas erradas'
>>> p.entra('6666666', 9, 27)
2
>>> p.entra('7777777', 9, 27)
3
>>> p.entra('8888888', 9, 27)
'entra: piscina cheia'
>>> p.ocupacao()
3
>>> p.sai('7777777', 10, 37)
'valor a pagar: Euro 3.0'
>>> p.sai('5555555', 17, 35)

class piscina:
    def __init__(self, places):
        self.places = places
        self.reg = {}

    def ocupacao(self):
        return len(self.reg)

    def entra(self, bi, h, m):
        if self.ocupacao() == self.places:
            raise ValueError('entra: piscina cheia')
        if bi in self.reg:
            raise ValueError('entra: a pessoa esta na piscina')
        if h not in range(0, 24) or m not in range(0,60):
            raise ValueError('entra: horas erradas')
        self.reg[bi] = h*60+m
        return self.ocupacao()

    def sai(self, bi, h, m):
        if self.ocupacao() == 0:
            raise ValueError('sai: vazia')
        if bi not in self.reg:
            raise ValueError('sai: a pessoa nao esta na piscina')
        if h not in range(0, 24) or m not in range(0,60) \
           or (60*h+m < self.reg[bi]):
            raise ValueError('entra: horas erradas')
        t = 60*h+m - self.reg[bi]
        val = 2 + (t//60 + (1 if t%60!=0 else 0))*0.5
        val = val if val <5 else 5
        del self.reg[bi]
        return 'valor a pagar: Euro {}'.format(val if val < 5 else 5.0)
```

# Pergunta 11 (val. 1.5)

Usando palavras suas e, no máximo, em três linhas responda às seguintes questões. Respostas dadas através de exemplos serão classificadas com zero valores.

**a)** Diga o que é um algoritmo e quais são as suas características (sem as descrever).

Um algoritmo é uma sequência finita de instruções bem definidas e não ambíguas cada uma das quais pode ser executada mecanicamente num período de tempo finito e com uma quantidade de esforço finito. Um algoritmo é rigoroso, eficaz e deve terminar.

**b)** Qual a relação entre um programa e um algoritmo?

Um programa corresponde a um algoritmo escrito numa linguagem de programação.

**c)** Qual a relação entre um processo e um programa.

Um processo corresponde ao conjunto de ações tomadas por um computador durante a execução de um programa.