TÉCNICO LISBOA

Fundamentos da Programação

Exame

24 de Novembro de 2021

13:00–15:00

Nome: _____ Número: _____

- Esta prova, individual e sem consulta, tem **11** páginas com **9** perguntas e uma duração de 2 horas. A cotação de cada pergunta está assinalada entre parêntesis.
- Em cima da mesa devem apenas estar o enunciado, caneta ou lápis e borracha e cartão de aluno. Não é permitida a utilização de folhas de rascunho, telemóveis, calculadoras, etc.
- Antes do início do tempo, dispõem de 5 minutos adicionais para ler o enunciado e identificar as folhas da prova.
- Escreva o seu número em todas as folhas da prova legível e a caneta.
- O tamanho das respostas deve ser limitado ao espaço fornecido para cada questão. O corpo docente reserva-se o direito de não considerar a parte das respostas que excedam o espaço indicado. A última página (em branco) pode ser usada para rascunho, não se aceitando respostas nesta página.
- Pode responder utilizando lápis.
- Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos.
- Boa sorte!

Pergunta	Cotação	Nota
1.	3.0	
2.	3.0	
3.	1.0	
4.	1.0	
5.	3.0	
6.	2.5	
7.	2.0	
8.	3.0	
9.	1.5	
Total	20.0	

1. (3.0) Para cada uma das seguintes afirmações, diga se é verdadeira (V) ou falsa (F). Cada resposta correta vale 0.3 valores e *cada resposta errada desconta 0.3 valores*.

   (a) O Python é uma linguagem compilada.
   Resposta: ___
   Resposta:
   F

   (b) Um algoritmo pode demorar um tempo infinito a executar.
   Resposta: ___
   Resposta:
   F

   (c) As gramáticas BNF são úteis para representar a semântica de uma linguagem.
   Resposta: ___
   Resposta:
   F

   (d) Uma exceção lançada dentro de uma função interrompe imediatamente a função.
   Resposta: ___
   Resposta:
   V

   (e) Se `t` é o tuplo `(1, [2], [3])`, então `t[1][0] = 3` é uma operação válida.
   Resposta: ___
   Resposta:
   V

   (f) Assumindo que `s` corresponde a uma variável de tipo *string*, a operação `s = s[-1]` levantará sempre um erro.
   Resposta: ___
   Resposta:
   F

   (g) Sendo `d` um dicionário vazio, `d[0] += 1` gera sempre um erro de *runtime*.
   Resposta: ___
   Resposta:
   V

   (h) Um algoritmo de complexidade $O(1)$ realiza o mesmo número de operações independentemente do tamanho da entrada.
   Resposta: ___
   Resposta:
   V

   (i) Para aceder ao valor de uma variável global no corpo de uma função precisamos utilizar a instrução `global`.
   Resposta: ___
   Resposta:
   F

(j) Em Python é possível passar funções como parâmetros de uma função e devolver funções como valor de uma função.

Resposta: ___

Resposta:

V

2. Considere a linguagem cujas frases começam por um ou mais símbolos `<`, o qual é seguido por uma ou mais ocorrências do símbolo `a`, seguido de uma ou mais ocorrências do símbolo `b`, seguido de uma ou mais ocorrências do símbolo `c`, após o que terminam com o mesmo número de símbolos `>` que `<` do início. Por exemplo, `<<<abc>>>` e `<aabcccc>` são frases da linguagem; `<cab>`, `<ab>` e `<<abc>` não o são.

(a) (1.0) Escreva uma gramática em notação BNF para a linguagem descrita.

Resposta:

⟨S⟩ ::= < ⟨S⟩ > | < ⟨Meio⟩ >
⟨Meio⟩ ::= ⟨A⟩ ⟨B⟩ ⟨C⟩
⟨A⟩ ::= ⟨A⟩ `a` | `a`
⟨B⟩ ::= ⟨B⟩ `b` | `b`
⟨C⟩ ::= ⟨C⟩ `c` | `c`

(b) (0.5) Indique o símbolo inicial, os símbolos não terminais e os símbolos terminais da sua gramática.

Resposta:

Símbolo inicial: S
Símbolos não terminais: S, Meio, A, B, C
Símbolos terminais: `<`, `>`, `a`, `b`, `c`

(c) (1.5) 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 carateres.

Resposta:

```
def reconhece(cad):
    if isinstance(cad, str):
        symb_open = cad.count('<')
        symb_close = cad.count('>')
        symb_a = cad.count('a')
        symb_b = cad.count('b')
        symb_c = cad.count('c')
        symb = symb_open*'<' + symb_a*'a' + symb_b*'b' \
        + symb_c*'c' +  symb_close*'>'

        return symb_open == symb_close and symb_open >= 1 \
        and symb_a >= 1 and symb_b >= 1 and symb_c >= 1 \
        and cad == symb

    raise ValueError("Argumento nao eh string")
```

3. (1.0) Escreva a função `combina_tuplos` que recebe dois tuplos do mesmo comprimento, `t1` e `t2`, e devolve um tuplo de pares em que o 1º elemento pertence a `t1` e o 2º elemento pertence a `t2`. Os elementos de `t1` nos pares aparecem pela mesma ordem que em `t1`, e os elementos de `t2` nos pares aparecem pela ordem inversa à ordem em `t2`. Por exemplo,

```
>>> combina_tuplos((1, 2, 3, 4), ('a', 'b', 'c', 'd'))
((1, 'd'), (2, 'c'), (3, 'b'), (4, 'a'))
```

Resposta:

```
def combina_tuplos(t1, t2):
    res = ()
    for i in range(len(t1)):
        res += ((t1[i], t2[-i-1]),)
    return res
```

4. (1.0) Escreva a função `procura_somandos` que recebe uma lista de números naturais `lst` e um natural `num` e devolve verdadeiro se existirem dois números, em posições diferentes da lista, cuja soma é igual a `num`, ou falso em caso contrário. Por exemplo,

```
>>> procura_somandos([1,2,3,4], 2)
False
>>> procura_somandos([1,2,3,4], 5)
True
>>> procura_somandos([1,1], 2)
True
```

Resposta:

```
def procura_somandos(lst, num):
    for i in range(len(lst)):
        procura = num - lst[i]
        if procura in lst[i+1:]:
            return True
    return False
```

5. Considere o TAD `tempo`, constituído por uma hora (inteiro entre 0 e 23) e minutos (inteiro entre 0 e 59). O TAD `tempo` dispõe das seguintes operações básicas:

   - *Construtor*:
     - $cria\_tempo : inteiro \times inteiro \mapsto tempo$
       $cria\_tempo(h, m)$ tem como valor o tempo de hora h e minutos m.
   - *Seletores*:
     - $hora\_tempo : tempo \mapsto inteiro$
       $hora\_tempo(t)$ tem como valor a hora do tempo t.
     - $minutos\_tempo : tempo \mapsto inteiro$
       $minutos\_tempo(t)$ tem como valor os minutos do tempo t.
   - *Reconhecedor*:

  - $eh\_tempo : universal \mapsto lógico$
    $eh\_tempo(arg)$ tem valor $verdadeiro$ apenas se arg é um tempo.

- *Testes*:
  - $tempos\_iguais : tempo \times tempo \mapsto lógico$
    $tempos\_iguais(t1, t2)$ tem valor $verdadeiro$ apenas se $t1$ e $t2$ são tempos iguais.
  - $depois : tempo \times tempo \mapsto lógico$
    $depois(t1, t2)$ tem valor $verdadeiro$ apenas se $t1$ é posterior a $t2$.

(a) (0.25) Escolha uma representação interna para o tipo `tempo`.

(b) (1.75) Implemente as operações básicas para a representação escolhida. Apenas o construtor necessita de verificar a validade dos argumentos. Se os argumentos não forem válidos o construtor deve gerar um erro do tipo `ValueError`, com a mensagem `'cria_tempo: argumentos inválidos'`.

Resposta:

```
# usando como representação interna (hora, minutos)
def cria_tempo(hora, mins):
    if not valida_hora_minutos(hora, mins):
        raise ValueError('cria_tempo: argumentos invalidos')
    return (hora, mins)

def valida_hora_minutos(hora, mins):
    return type(hora) == int and 0<= hora <= 23 and\
           type(mins) == int and 0<= mins <= 59

def hora_tempo(tmp):
    return tmp[0]

def minutos_tempo(tmp):
    return tmp[1]

def eh_tempo(x):
    return type(x) == tuple and len(x) == 2 and \
    valida_hora_minutos(x[0], x[1])

def tempos_iguais(t1, t2):
    return t1 == t2

def depois(t1,t2):
    return hora_tempo(t1) > hora_tempo(t2) or\
           hora_tempo(t1) == hora_tempo(t2) and \
           minutos_tempo(t1) > minutos_tempo(t2)
```

(c) (1.0) Implemente a operação de **alto nível** `diferenca_tempos(t1, t2)`, em que `t1` é um `tempo` posterior a `t2`, que devolve um `tempo` correspondente à diferença entre `t1` e `t2`. Não precisa de validar os argumentos. Por exemplo,

```
>>> t1 = cria_tempo(13, 45)
>>> t2 = cria_tempo(11, 50)
>>> dif = diferenca_tempos(t1, t2)
```

```
>>> hora_tempo(dif)
1
>>> minutos_tempo(dif)
55
```

Resposta:

```
def diferenca_tempos(t1,t2):
    t1_mins = hora_tempo(t1) * 60 + minutos_tempo(t1)
    t2_mins = hora_tempo(t2) * 60 + minutos_tempo(t2)
    dif = t1_mins - t2_mins
    dif_horas = dif // 60
    dif_mins = dif % 60
    return cria_tempo(dif_horas, dif_mins)
```

6. Nesta questão considere que foi definido o TAD `tempo` (ver pergunta 5).

   Considere que a informação sobre o preço de aluguer de bicicletas é representada por um dicionário em que as chaves são naturais correspondentes ao tipo de bicicleta e os valores os preços à hora do aluguer. Por exemplo,

   ```
   info_bics = {1: 2.5, 2 : 3}
   ```

   indica que as bicicletas de tipo 1 são alugadas a 2.5 euros à hora, e as de tipo 2 a 3 euros à hora. Considere agora que a informação sobre as bicicletas de uma empresa de aluguer de bicicletas é guardada num dicionário em que as chaves são naturais (correspondentes a identificadores únicos das bicicletas da empresa). A cada chave está associado um dicionário de chaves `'tipo'` e `'estado'`. O valor associado à chave `'tipo'` é o tipo da bicicleta, e o valor associado à chave `'estado'` pode ser `'livre'` se a bicicleta estiver no parque, ou um elemento do tipo `tempo` (tal como definido na pergunta 5) indicando a hora de início do aluguer, caso contrário. Por exemplo,

   ```
   parque_bics = {100: {'tipo': 1, 'estado': 'livre'},
                  101: {'tipo': 1, 'estado': cria_tempo(13, 0)},
                  200: {'tipo': 2, 'estado': 'livre'},
                  201: {'tipo': 2, 'estado': 'livre'}}
   ```

   (a) (1.0) Defina a função `disponiveis_tipo(parque_bics, tipo)` que recebe um dicionário com a informação sobre as bicicletas de uma empresa e um natural correspondendo a um tipo de bicicleta, `tipo`, e devolve uma lista com os identificadores das bicicletas de tipo `tipo` disponíveis para alugar. Por exemplo, tendo em atenção as atribuições anteriores:

   ```
   >>> disponiveis_tipo(parque_bics, 1)
   [100]
   >>> disponiveis_tipo(parque_bics, 2)
   [200, 201]
   ```

   Resposta:

```
def disponiveis_tipo(parque_bics, tipo):
    return [num for num in parque_bics
            if parque_bics[num]['tipo'] == tipo and
            parque_bics[num]['estado'] == 'livre']
```

(b) (1.5) Defina a função `entrega_bic(parque_bics, info_bics, num, hora_ent)` que recebe um dicionário com a informação sobre as bicicletas de uma empresa, `parque_bics`, um dicionário com a informação sobre os preços de aluguer, `info_bics`, um natural correspondendo a um número de bicicleta, `num`, e um elemento do tipo `tempo`, `hora_ent`, e devolve o preço a pagar pelo aluguer, atualizando a informação em `parque_bics`. O tempo a cobrar é sempre arredondado para as horas acima. Por exemplo, se o tempo de aluguer for 1 hora e 50 minutos serão cobradas 2 horas.

Considere que a chave `num` existe sempre no dicionário `parque_bics`. Se a bicicleta de número `num` se encontrar no parque, a sua função deve escrever a mensagem "Bicicleta no parque". Se a hora de entrega `hora_ent` não for posterior à hora de início do aluguer, a sua função deve escrever a mensagem "Hora errada". Por exemplo, tendo em atenção as atribuições anteriores:

```
>>> entrega_bic(parque_bics, info_bics, 100, cria_tempo(12,0))
Bicicleta no parque
>>> entrega_bic(parque_bics, info_bics, 101, cria_tempo(12,50))
Hora errada
>>> entrega_bic(parque_bics, info_bics, 101, cria_tempo(13,50))
2.5
>>> parque_bics
{100: {'tipo': 1, 'estado': 'livre'},
 101: {'tipo': 1, 'estado': 'livre'},
 200: {'tipo': 2, 'estado': 'livre'},
 201: {'tipo': 2, 'estado': 'livre'}}
```

Resposta:

```
def entrega_bic(parque_bics, info_bics, num, hora_ent):
    bic = parque_bics[num]
    if bic['estado'] == 'livre':
        print('Bicicleta no parque')
    elif not depois(hora_ent, bic['estado']):
        print('Hora errada')
    else:
        tempo_uso = diferenca_tempos(hora_ent, bic['estado'])
        horas_uso = hora_tempo(tempo_uso)
        horas_a_cobrar = horas_uso if minutos_tempo(tempo_uso) == 0 \
        else horas_uso + 1
        bic['estado'] = 'livre'
        return horas_a_cobrar * info_bics[bic['tipo']]
```

7. Usando um ou mais dos funcionais sobre listas (`filtra`, `transforma`, `acumula`), escreva as funções seguintes. As suas funções devem conter apenas uma instrução, a instrução `return`. Não é necessário validar os dados de entrada.

(a) (1.0) A função `norma_euclidiana`, que recebe uma vector de inteiros e devolve a sua norma euclidiana:

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}}$$

Por exemplo:

```
>>>> norma_euclidiana((4, 3))
5
```

Resposta:

```
def norma_euclidiana(v):
    return (acumula(lambda x,y: x+y, \
                    transforma(lambda x:x*x, v)))**0.5
```

(b) (1.0) A função `produto_multiplos`, que recebe um inteiro positivo `n` e outro inteiro positivo `d` menor ou igual que `n`, e devolve o produto de todos os números entre `1` e `n` que são múltiplos de `d`. Por exemplo:

```
>>> produto_multiplos(10, 3)
162
```

Resposta:

```
def produto_multiplos(n, d):
    return acumula(lambda x,y:x*y, \
                    filtra(lambda x:x%d==0, range(1, n+1)))
```

8. Considere uma função `agrupa()` que recebe dois tuplos e retorna um novo tuplo formado por tuplos de 2 elementos, em que no primeiro elemento do tuplo na posição i será o elemento na posição i do primeiro tuplo de entrada e o segundo elemento o elemento na posição i do segundo tuplo de entrada. Caso um dos tuplos seja mais pequeno, o elemento correspondente a este tuplo deverá ficar com o valor 0. Por exemplo:

```
>>> agrupa((1,4,9), (2,3,5,7,8))
((1,2), (4,3), (9,5), (0,7), (0,8))
```

(a) (1.0) Escreva esta função usando uma iteração linear.

Resposta:

```
def agrupa(t1, t2):
    res = ()
    l1, l2 = len(t1), len(t2)
    m = max(l1, l2)
    for i in range(m):
        a = t1[i] if i < l1 else 0
        b = t2[i] if i < l2 else 0
        res += ((a, b),)
    return res
```

(b) (1.0) Escreva esta função usando uma recursão linear de operações adiadas.

Resposta:

```
def agrupa(t1, t2):
    if len(t1) == 0:
        if len(t2) == 0:
            return ()
        else:
            return ((0, t2[0]),) + agrupa((), t2[1:])
    else:
        if len(t2) == 0:
            return ((t1[0], 0),) + agrupa(t1[1:], ())
        else:
            return ((t1[0], t2[0]),) + agrupa(t1[1:], t2[1:])
```

(c) (1.0) Escreva esta função usando uma recursão linear de cauda.

Resposta:

```
def merge(t1, t2):
    def aux(ta, tb, res):
        if len(ta) == 0:
            if len(tb) == 0:
                return res
            else:
                return aux((), tb[1:], res + ((0, tb[0]),))
        else:
            if len(tb) == 0:
                return aux(ta[1:], (), res + ((ta[0], 0),))
            else:
                return aux(ta[1:], tb[1:], res + ((ta[0], tb[0]),))

    return aux(t1, t2, ())
```

9. (1.5) Escreva uma função `fconta()` que recebe duas *strings*, representando o nome de dois ficheiros. Esta função deverá criar um ficheiro com o nome da segunda *string* em que em cada linha deve constar o número de carateres da linha correspondente do ficheiro com o nome da primeira *string*.

   Resposta:

```
def fconta(s1, s2):
    with open(s1, 'r') as f1:
        linhas = f1.readlines()
    with open(s2, 'w') as f2:
        for l in linhas:
            f2.write(str(len(l) - 1))
            f2.write('\n')
```