Prova Final de Linguagens de Programação
- DCC024 -
Ciência da Computação e Sistemas de Informação

Ao concordar participar da prova, você dá sua palavra de honra que suas respostas são fruto único do seu trabalho. Você pode consultar a internet, por exemplo, mas não pode consultar outros seres vivos para fazer a prova.

As regras do jogo:

- Você pode consultar entidades inanimadas, mas não pode consultar entidades animadas.
- Você precisa enviar três arquivos para `fernando@dcc.ufmg.br` até as 19h00 do dia 30 de Janeiro, a saber:
  - `sol.sml`: solução da questão 1.
  - `sol.py`: solução da questão 2.
  - `sol.pl`: solução da questão 3.
- Seu e-mail deve ter o título `Prova_DCC024`. Não use qualquer outro título.
- O único texto no corpo de seu e-mail deve ser seu nome, e, possivelmente, a resposta da questão extra. Não escreva o código dos arquivos no e-mail: envie os arquivos como anexos.
- Caso você queira reenviar suas respostas, simplesmente responda a mensagem de seu e-mail anterior. Não envie e-mails separados!
- Não há como tirar dúvidas durante a prova. Lembre-se da cláusula sobre entidades animadas. Isso inclui escrever e-mails para o instrutor.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Extra 0.5

Questão extra (0.5): De qual livro foi retirado o trecho abaixo?

1. (10 Pontos) Nesta questão, você deverá implementar uma função `group`, em SML. Essa função possui o seguinte tipo: `int list -> int list -> (int * int) list`. Ela recebe duas listas de inteiros `means` e `points`. Ela então retorna uma lista de pares de inteiros, `(m, p)`, tal que:

   - `m` é um elemento de `means`;
   - `p` é um elemento de `points`;
   - não existe outro elemento `m'` em `means` tal que a diferença absoluta entre `m'` e `p` seja menor que a diferença absoluta entre `m` e `p`.

   Alguns exemplos da execução da função `group` podem ser vistos logo abaixo:

```
- group [2] [1, 3, 4, 6, 7, 9, 10];
val it = [(2,1),(2,3),(2,4),(2,6),(2,7),(2,9),(2,10)] : (int * int) list

- group [2, 5, 8] [1, 3, 4, 6, 7, 9, 10];
val it = [(2,1),(2,3),(5,4),(5,6),(8,7),(8,9),(8,10)] : (int * int) list

- group [1, 8] [2, 3, 4, 5, 6, 7];
val it = [(1,2),(1,3),(1,4),(8,5),(8,6),(8,7)] : (int * int) list

- group [1, 8] nil;
val it = [] : (int * int) list

- group nil [2, 3, 4, 5, 6, 7];
uncaught exception Match [nonexhaustive match failure]
```

   Você não precisa tratar o caso em que a lista `means` esteja vazia. Em outras palavras, deixe que a ocorrência dessa lista vazia como parâmetro de entrada cause uma exceção de casamento não exaustivo.

2. Esta questão refere-se à classe abaixo, que cria números inteiros a partir de seus dígitos, organizados em uma lista:

```
class Num:
  def __init__(self, digits):
    self.n = 0
    pow = 1
    for d in range(len(digits)):
      digit = digits[len(digits) - 1 - d]
      if digit < 0 or digit > 9:
        raise ArithmeticError('Isso parece um digito? ' + str(digit))
      self.n += pow * digit
      pow *= 10
  def __str__(self):
    return str(self.n)
```

Exemplos de uso seguem abaixo:

```
>>> n1 = Num([1, 2, 3]); print(n1)
123
>>> n1 = Num([0, 1, 2, 3]); print(n1)
123
>>> n1 = Num([0, 1, 2, 3, 0]); print(n1)
1230
>>> n1 = Num([10, 2, 3])
ArithmeticError: Isso parece um digito para voce? 10
```

(a) (5 Pontos) Implemente uma classe `Neg`, que estenda a classe `Num`. Toda instância de `Neg` representa um número negativo. Por exemplo:

```
>>> n1 = Neg([1, 2, 3]); print(n1)
-123
>>> n1 = Neg([0, 1, 2, 3]); print(n1)
-123
>>> n1 = Neg([0, 1, 2, 3, 0]); print(n1)
-1230
```

(b) (5 Pontos) Implemente uma classe `Dec`, que estenda a classe `Num`, de forma que toda instância de `Dec` represente um número decimal. Note que o construtor de `Dec` recebe um parâmetro extra:

```
>>> n1 = Dec([1, 2, 3], [1, 2, 3]); print(n1)
123.123
>>> n1 = Dec([1, 2, 3], [0, 1, 2, 3]); print(n1)
123.0123
>>> n1 = Dec([1, 2, 3], [0, 1, 2, 3, 0]); print(n1)
123.0123
>>> n1 = Dec([0, 1, 2, 3], [0, 1, 2, 3, 0]); print(n1)
123.0123
>>> n1 = Dec([0, 1, 2, 3, 0], [0, 1, 2, 3, 0]); print(n1)
1230.0123
```

3. Nesta questão você deverá trabalhar com a noção de *distância de edição*. A distância de edição entre duas cadeias de caracteres, s_1 e s_2, e o menor número de caracteres que você pode inserir, remover ou modificar em s_1 até obter s_2.

   (a) (4 Pontos) Escreva um predicado `hop_one(L, S, D)`, que seja verdade se `D` for uma lista produzida pela inserção de um caractere presente na lista `L` em `S`. Exemplos:

   ```
   ?- hop_one([a, b, c], [x, y, z], [x, a, y, z]).
   true ;
   false.

   ?- hop_one([a, b, c], [x], [x, Var]).
   Var = a ;
   Var = b ;
   Var = c ;
   false.
   ```

   (b) (4 Pontos) Escreva um predicado `replace(L, S, D)`, que seja verdade se for possível transformar `D` em `S` substituindo-se um caractere de `D` por um caractere presente em `L`. Por exemplo:

   ```
   ?- replace([a, b, c], [x, y, a], [x, y, z]). # Basta substituir 'z' por 'a'
   true ;
   false.

   ?- replace([a, b, c], [x, y, z], [x, y, a]).
   false.

   ?- replace([a, b, c], [x, y, a], [x, Var, z]).
   Var = y ;
   false.
   ```

   (c) (2 Pontos) Finalmente, escreva um predicado `hop_two(L, S, D)`, que seja verdade se for possível transformar `S` em `D` usando primeiro uma chamada a `hop_one`, e depois, uma chamada a `replace`. Por exemplo:

   ```
   ?- hop_two([a, b, c], [x, y, b], [x, a, y, z]).
   true ; # Insere 'a' após 'x', e depois transforma 'b' em 'z'.
   false.

   ?- hop_two([a], [x, y, a], [x, y, a, z]).
   true ; # insere 'a' na frente de 'x', depois transforma 'x' (lista 3) em 'a'.
   true ; # insere 'a' depois de 'a' e depois transforma 'z' em 'a'.
   false.
   ```