Segundo Trabalho de Programação Funcional – 2020/1

Data de entrega: 25/05/2021 (até 23:55)

Valor: 22 pontos

Obs1: Esse trabalho pode ser desenvolvido em **Dupla** ou **Individual,** mas a arguição na apresentação e a respectiva nota será individual.

Obs2: A nota será dada pela nota de desenvolvimento (de 0 a 20) multiplicada pela nota de arguição (de 0 a 1), onde o aluno deverá demonstrar completo entendimento do trabalho desenvolvido.

PARTE A – Algoritmos de ordenação

Para os exercícios de ordenação, considere as 14 listas a seguir como exemplos para teste dos diversos algoritmos:

l1=[1..2000]

l2=[2000,1999..1]

l3=l1++[0]

l4=[0]++l2

l5=l1++[0]++l2

l6=l2++[0]++l1

l7=l2++[0]++l2

x1=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

x2=[20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]

x3=[11,12,13,14,15,16,17,18,19,20,1,2,3,4,5,6,7,8,9,10]

x4=[10,9,8,7,6,5,4,3,2,1,20,19,18,17,16,15,14,13,12,11]

x5=[11,12,13,14,15,5,4,3,2,1,16,17,18,19,20,10,9,8,7,6]

x6=[1,12,3,14,5,15,4,13,2,11,6,17,8,19,20,10,9,18,7,16]

x7 = [20,8,2,11,13,3,7,18,14,4,16,10,15,1,9,17,19,12,5,6]

Exercício 1) Em relação ao algoritmo de ordenação **Bolha,** faça as implementações em Haskell solicitadas abaixo**. Em todas as variações abaixo, incluindo o código original, você deve incluir um contador de trocas realizadas durante a ordenação.**

- Original Bolha: implementação do código visto em aula.

- Variação 1: parada do algoritmo é antecipada quando uma iteração de trocas é finalizada sem que nenhuma troca efetiva seja realizada na iteração completa.

- Variação 2: faz parada antecipada e, a cada iteração de trocas, a avaliação é realizada desconsiderando-se o último elemento cuja posição foi fixada. Ou seja, diminui o tamanho da lista a ser ordenada a cada iteração.

- Realizar execuções comparativas nas listas dadas como exemplo entre os algoritmos das variações e do algoritmo original para avaliar: 1) o número de **trocas** em cada execução; 2) o tempo de execução (apenas verifiquem se existe uma mudança aparente de tempo de processamento). Apresentar esses resultados em um arquivo PDF, fazendo uma análise dos resultados encontrados.

Exercício 2) Em relação ao algoritmo de ordenação **Selecao,** faça as implementações em Haskell solicitadas abaixo. **Em todas as variações abaixo, incluindo o código original, você deve incluir um contador de trocas realizadas durante a ordenação.**

- Original Seleção: implementação do código visto em aula.

- Variação1: Refaça o código original para que a busca pelo menor elemento (função mínimo) e a eliminação desse menor elemento da lista a ser ordenada (função remove) ocorra numa mesma função (remove_menor), sem a necessidade de se percorrer a lista duas vezes a cada iteração.

- Variação 2: Refaça a implementação do algoritmo Seleção usando funções genéricas (foldr ou foldr1) .

- Realizar execuções comparativas nas listas dadas como exemplo entre os algoritmos das variações e do algoritmo original para avaliar: 1) o número de **trocas** em cada execução; 2) o tempo de execução (apenas verifiquem se existe uma mudança aparente de tempo de processamento). Apresentar esses resultados em um arquivo PDF, fazendo uma análise dos resultados encontrados.

Exercício 3) Em relação ao algoritmo de ordenação **Inserção,** faça as implementações em Haskell solicitadas abaixo. **Em todas as variações abaixo, incluindo o código original, você deve incluir um contador de comparações realizadas durante a ordenação.**

- Original Inserção: implementação do código visto em aula.

    - Variação 1: Refaça a implementação do algoritmo Inserção usando funções genéricas (foldr ou foldr1).

- Realizar execuções comparativas nas listas dadas como exemplo entre os algoritmos das variações e do algoritmo original para avaliar: 1) o número de **trocas** em cada execução; 2) o tempo de execução (apenas verifiquem se existe uma mudança aparente de tempo de

processamento). Apresentar esses resultados em um arquivo PDF, fazendo uma análise dos resultados encontrados.

Exercício 4) Em relação ao algoritmo de ordenação **quicksort** visto em sala de aula e laboratório. **Em todas as variações abaixo, incluindo o código original, você deve incluir um contador de comparações realizadas durante a ordenação.**

- Original Quicksort: implementação do código visto em aula.

- Variação 1: modifique o algoritmo original para que ao invés dos elementos maiores e menores serem encontrados com buscas independentes, que seja elaborada e utilizada a função **divide** que percorre a lista uma única vez, retornando os elementos menores em uma lista e os elementos menores em outra.

EX: > divide 'j' "pindamonhangaba" Resposta: ("idahagaba","pnmonn")

- Variação 2: modifique a variação 1 para que o elemento pivô seja obtido a partir da análise dos 3 primeiros elementos da lista, sendo que o pivô será o elemento mediano entre eles. Exemplo: na lista [3, 9, 4, 7, 8, 1, 2], os elementos 3, 9 e 4 seriam analisados e o pivô escolhido seria 4. Caso a lista a ser analisada tenha menos que 3 elementos, o pivô é sempre o primeiro.

- Realizar execuções comparativas nas listas dadas como exemplo entre os algoritmos da Variação 1, 2 e do algoritmo original para avaliar: 1) o número de comparações em cada execução; 2) o tempo de execução (apenas verifiquem se existe uma mudança aparente de tempo de processamento). Apresentar esses resultados em um arquivo PDF, fazendo uma análise dos resultados encontrados.

Exercício 5)

- Pesquise e implemente em Haskell o algoritmo mergesort, incluindo um contador de comparações elementares realizadas durante a ordenação.

- Realizar execuções comparativas nas listas dadas como exemplo entre os algoritmos selecionados como as melhores versões do Seleção e Quicksort, além do algoritmo Mergesort, para avaliar: 1) o número de comparações em cada execução; 2) o tempo de execução (apenas verifiquem se existe uma mudança aparente de tempo de processamento). Apresentar esses resultados em um arquivo PDF, fazendo uma análise dos resultados encontrados.

Obs: as análises solicitadas nos exercícios 1, 2, 3, 4 e 5, devem ser entregues em um único arquivo PDF, que deve ser enviado junto com os códigos dos exercícios.

PARTE B – Tipos Algébricos e Ávores

Exercício 6) Dada a definição de tipos abaixo, similar à vista em aula.

```
data Exp =
  Val Int -- um numero
| Add Exp Exp -- soma de duas expressoes
| Sub Exp Exp -- subtração de duas expressoes

avalia :: Num a => Exp a -> a
avalia (Val x) = x
avalia (Add exp1 exp2) = (avalia exp1) + (avalia exp2)
avalia (Sub exp1 exp2) = (avalia exp1) - (avalia exp2)
```

a) Expanda as definições acima para que sejam incluídas expressões usando as operações multiplicação e divisão (além de soma e subtração).

b) Avalie as expressões abaixo, primeiro declarando-as de acordo com a sintaxe do tipo algébrico e depois executando a função avalia sobre essas declarações:

(3+12)*(15-5)/(1*3) e - ((6+8-5+1)*(2+6/2))

Exercício 7) Jogo "Pedra, Papel, Tesoura"

a) Defina um tipo algébrico **Jogada**, para representar uma possível jogada, ou seja: Pedra ou Papel ou Tesoura.

b) Defina a função **vence**, que recebe duas instâncias de Jogada **j1** e **j2** e retorna True se **j1** vence **j2** no jogo. Ex:

> Pedra 'vence' Tesoura

True

> Tesoura 'vence' Pedra

False

> Tesoura 'vence' Tesoura

False

b) Defina a função **vencedoras** que recebe uma lista de duplas de jogadas e retorna uma lista com as jogadas vencedoras. No caso de uma dupla que resulte em empate, retorna o primeiro elemento da tupla. Ex:

> vencedoras [(Pedra,Tesoura),(Tesoura,Pedra), (Tesoura,Tesoura)]

[Pedra, Pedra, Tesoura]

Exercício 8) Lógica Nebulosa

a) Crie o tipo algébrico **Nebuloso** que pode ter os valores *Verdadeiro*; *Falso*; ou *Talvez* que define um conceito intermediário entre Verdadeiro e Falso. No caso do valor Talvez, ele deve ser associado a ele um valor float que representa o grau de pertinência. O valores Verdadeiro e Falso não devem estar associados a valores.

b) Crie uma função **fuzzifica** que recebe um valor numérico (Float) representando o grau de pertinência e retorna um conceito Nebuloso, com o valor Falso caso o grau seja menor ou igual a 0, Verdadeiro se for maior ou igual a 1 e Talvez associado ao grau de pertinência, caso contrário.

c) Considere a definição **nebulosa** do conceito "pessoa alta" como: dada a altura da pessoa , então pertinência a pessoa alta é dada por (altura-1,70)/0,20

Cria a função **verifica_alto** que recebe um valor que representa a altura de uma pessoa (em float) e retorna o valor nebuloso associando a altura ao conceito pessoa alta.

d) Considere a definição **nebulosa** do conceito "carro barato" como: dado o custo de um carro, então pertinência a carro barato é dado por: (50.000-Custo)/ 20.000

Cria a função **verifica_barato** que recebe um valor que representa o custo de um carro (em float) e retorna o valor nebuloso associando o custo ao conceito carro barato.

Exercício 9) a) Defina o Tipo Algébrico Estudante para registrar informações de alunos de universidades e colégios. O objetivo é realizar a união das informações destes estudantes. Cada ocorrência de estudante deve ter um identificador que diferencie universitários de colegiais. Além disso, se for um aluno de colégio, devem existir as informações: ano (1º, 2º ou 3º), nome do colégio (Nacional, Olimpo e Gabarito) matricula (5 letras), altura e peso. Se for um aluno universitário, devem existir as informações: nome da universidade (UFU, UNITRI e UNA), nome do curso (Computação, Medicina, Direito e Música), matricula (8 dígitos), altura e idade. Crie uma pequena base de estudantes com pelo menos 20 instâncias, sendo 10 alunos colegiais e 10 alunos universitários. Em cada grupo, 3 alunos devem ter altura abaixo de 1,70, 5 alunos devem ter altura entre 1,70 e 1,90 e 2 alunos devem ter altura acima de 1,90.

b) Implemente a função descobre_altos, utilizando a função verifica_alto do exercício anterior para receber a base de estudantes (ou seja, a lista contendo os dados dos 2º estudantes armazenados) e retornar uma lista de duplas com o numero de matricula na 1ª posição da tupla, conceito nebuloso de pessoa alta (verdadeiro, falso ou talvez) na 2ª posição da tupla, ou seja, o retorno é do tipo (Int,Nebuloso)

Exercício 10) Considere o tipo algébrico **ArvBinInt** visto em sala para representar árvores binárias que armazenam números inteiros. Elabore as funções a seguir que manipulam árvores binárias:

folhas: recebe uma árvore binária e devolve uma listagem com todos os nós folhas (não internos) existentes na árvore. **Percorrer a árvore em pré-ordem.**
somaNosinternos: somar os valores de todos os elementos da árvore binária que são nós internos. **Percorrer a árvore em pós-ordem.**
pertence: recebe um valor inteiro e verifica se esse valor é igual a algum dos elementos da árvore binária. **Percorrer a árvore em ordem simétrica.**