IMD0029 - Estrutura de Dados Básicas 1 –2023.1 – Prova 04
Prof. Eiji Adachi M. Barbosa

Nome:_____
Matrícula: _____

ANTES DE COMEÇAR A PROVA, leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 10, pontos e o valor de cada questão é informado no seu enunciado.
- **Preze por respostas legíveis, bem organizadas e simples.**
- As respostas devem ser fornecidas preferencialmente em caneta. Respostas fornecidas a lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da prova do candidato (nota igual a zero).

ENTREGÁVEL

O entregável desta atividade deverá seguir a mesma estrutura de diretórios do código fonte que você recebeu com este enunciado, obviamente, contendo os arquivos fonte utilizados para construir sua solução nos diretórios de cada questão. Além disso, o diretório pai deverá ter o seu nome, seguindo o padrão `<PRIMEIRO_NOME>_<SOBRENOME>`. A estrutura de diretórios do entregável deverá ter uma estrutura similar a seguinte:

```
> JOAO_SILVA
   > q1
   > q2
   > q3
```

Toda esta estrutura de diretórios, incluindo os arquivos fonte com sua solução, deverá ser compactada num arquivo .zip (não é .rar, nem tar.gz, é zip!) que também deverá seguir o padrão `<PRIMEIRO_NOME>_<SOBRENOME>`. Este arquivo compactado deverá ser entregue via SIGAA até as **20:25**. **Este é um prazo fixo que não será estendido**, exceto em casos muito excepcionais (ex.: SIGAA fora do ar). Ou seja, entregas após este horário não serão aceitas. A atividade do SIGAA permite apenas um envio, portanto certifique-se de que está enviando a versão correta antes de anexar ao SIGAA.

Observação: Se você não implementou uma determinada questão, **NÃO coloque o diretório referente a essa questão no entregável (arquivo .zip).**

CRITÉRIOS DE CORREÇÃO

Para a correção desta atividade, serão levados em consideração, dentre outros, os seguintes pontos:

- Obediência às regras definidas para as assinaturas de função e para o entregável (arquivo .zip), conforme especificado no enunciado desta atividade
- Existência de erros ou warnings de compilação do código fonte
- Programas executam sem apresentar falhas e produzem os resultados esperados
- Soluções atendem critérios de complexidade, caso estabelecido no enunciado
- Apresentação e organização do código fonte entregue (identação, nome das variáveis, modularização do código em funções, etc)

Obs.: Para cada questão, já há uma estrutura de diretórios com um arquivo makefile e um pequeno teste executável, que pode ser acionado usando o comando "make run-test". Este é um teste simples que **não garante** a corretude da sua implementação. Ou seja, se sua implementação passou no teste executável disponibilizado junto a este enunciado, isto não é garantia de que ela está totalmente correta. Para fins de correção, eu utilizarei outra bateria de testes mais completa, além de analisar manualmente o código produzido.

Questão 1 (2,0 pontos): Sequências bitônicas inversas são aquelas que possuem duas sequências, sendo uma sequência inicial decrescente, seguida de uma sequência crescente. Ou seja, os elementos de uma sequência bitônica obedecem a seguinte relação:

$$A_0 > A_1 > \ldots > A_{i-1} > A_i < A_{i+1} < \ldots < A_n$$

Considere que um array bitônico-inverso é um array de inteiros sem repetições cujos elementos representam uma sequência bitônica inversa. Neste contexto, implemente umafunção que recebe como entrada um array bitônico-inverso e retorna o índice do elemento da "base". O elemento da "base" é o último elemento da sequência inicial decrescente e o primeiro elemento da sequência final crescente,ou seja, é o elemento Ai da relação acima. Sua função deverá obrigatoriamente ser **recursiva**, ter complexidade **Θ(lg(n))** e seguir a assinatura definida no arquivo SequenciaBitonica.h; sua implementação deverá ser feita no arquivo SequenciaBitonica.cpp.

Obs.: Nesta questão, não podem ser usadas instruções para realizar repetição, como for, while e do-while. Ou seja, você não poderá usar instruções de repetição; você deverá construir sua solução apenas com chamadas recursivas.

Questão 2 (2,0 pontos): Na notação infixa de expressões aritméticas, os operadores são escritos entre os operandos. Na notação pós-fixa, ou notação polonesa inversa, os operadores são escritos depois dos operandos. Eis alguns exemplos de expressões infixas e correspondentes expressões pós-fixas:

Infixa	Pós-fixa
1 + 2	1 2 +
2 / 2	2 2 /
1 + 2 * 3	1 2 3 * +
1 * 2 + 3 - 4	1 2 * 3 + 4 -

Nesta questão, implemente uma função que recebe como entrada um vector de strings representando uma expressão aritmética em notação pós-fixa e que retorna o seu valor numérico correspondente. A função está definida no arquivo "include/Expressao.h" e sua implementação deve ser feita no arquivo "src/Expressao.cpp". Sua solução deverá, obrigatoriamente, usar uma pilha; para isso, use a estrutura Stack da biblioteca padrão de C++. No arquivo "Expressao.cpp", existem exemplos de código de como declarar e manipular uma Stack de C++ e como converter uma string em um valor inteiro. Quaisquer dúvidas quanto ao uso dessa estrutura, pode me chamar.

Questão 3 (2,0 pontos): Para a estrutura de lista duplamente encadeada definida no arquivo ListaDuplamenteEncadeada.h., implemente uma função **iterativa** para remover os elementos repetidos da lista, retornando quantos elementos foram removidos. Caso existam elementos repetidos na lista, este método deve deixar apenas um destes elementos, removendo os demais. Ex.: A lista [2 → 1 → 1 → 1 → 5 → 5 → 0 → 7 → 7 → 8] após a execução do método deverá ficar igual a [2 → 1 → 5 → 0 → 7 → 8]. Sua implementação deverá ser feita no arquivo ListaDuplamenteEncadeada.cpp, respeitando a assinatura do método já existente.

Obs.: Nesta questão, assuma que a lista encadeada não está ordenada.

Questão 4 (2,0 pontos): Dada uma lista duplamente encadeada com sentinelas cabeça e cauda, implemente um método **iterativo** para inserir um elemento na lista, mantendo-a ordenada em ordem crescente. Considere que a lista ou está vazia ou já está ordenada em ordem crescente. Caso o elemento já exista na lista, a função não cria um

novo nó e retorna falso; caso o elemento ainda não exista na lista, deve-se criar um novo nó e encadeá-lo na lista, incrementando a quantidade de elementos da lista e mantendo-a ordenada.

Questão 5 (2,0 pontos): Considere uma tabela de dispersão que resolve colisões pelo método do endereçamento aberto com sondagem linear. Nesta questão, implemente o método de inserção desta tabela. Lembre-se que caso já exista um par na tabela com a chave que está sendo inserida, deve-se apenas atualizar o valor correspondente.

Obs.: Não é necessário se preocupar com os casos em que a tabela está cheia ou em que é necessário realizar o redimensionamento dinâmico da tabela.