Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC

Projeto 3 – Estruturas de Dados I
SCC-0202

Data de entrega: 01/12/2019

Observações

- Este projeto deverá ser realizado em duplas;
- Somente uma submissão por dupla é permitida;
- Especificar em um arquivo de texto junto com o código-fonte os nome e NUSP de cada um da dupla;
- O **código-fonte** e o **relatório** (em pdf) deve ser comprimidos em um único arquivo, no formato *zip*, com o nome "Projeto3.zip", e entregue via Atividades do TIDIA;
- O projeto deve ser desenvolvido usando somente a linguagem de programação C;
- Será utilizada uma ferramenta para verificar plágio nos códigos, sendo penalizados com nota 0 aqueles trabalhos que tenham plágio confirmado; Portanto, não serão permitidas cópias em hipótese alguma;
- O projeto será avaliado em termos de **corretude** e **legibilidade** do código;
- Em caso de dúvidas, envie email para eugenio.cabral@usp.br

Parte 1 - Implementação da Estrutura Coleção

Nota: 7 pontos

Uma **Coleção** é uma estrutura de dados genérica que permite o encapsulamento de outras estruturas tais como árvores e listas bastando apenas especificar o tipo da estrutura desejada. Uma Coleção é manipulada por meio de 5 operações básicas, e é composta pelo *tipo da estrutura* que será usada e *nó inicial*.

Tipo da estrutura

O tipo da estrutura que será encapsulada é definido durante a criação da Coleção, e **não deve** ser alterado durante a execução do programa. O tipo especificado determina como os valores serão inseridos e buscados na estrutura. Para este projeto serão considerados 5 tipos:

1. Lista ordenada;
2. Lista com inserção na última posição;
3. Lista com inserção na primeira posição;
4. Árvore binária de busca;
5. Árvore AVL.

Assim, em uma única implementação, temos uma estrutura flexível para comportar 3 variações de listas e 2 variações de árvores.

Nó inicial

Nos tipos de estruturas baseados em lista, o nó inicial é denominado de **cabeça**, e nos tipos baseados em árvore é denominado de **raiz**. Um **nó** armazena as seguintes informações:

1. Um inteiro que armazena o valor inserido;
2. Um ponteiro apontando para o nó à esquerda;
3. Um ponteiro apontando para o nó à direitta;
4. Um inteiro indicando a altura da árvore.

Nem todas informações armazenadas em um **nó** são utilizadas em todas as estruturas. Por exemplo, a **altura** só é relevante para estruturas baseadas em árvores, no caso de listas o valor deve ser sempre 0 (zero), afinal, todos nós da lista estão na mesma altura.

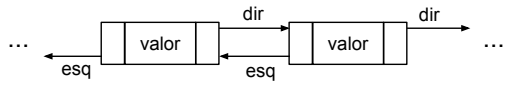

Figura 1: Ponteiros de nós em listas

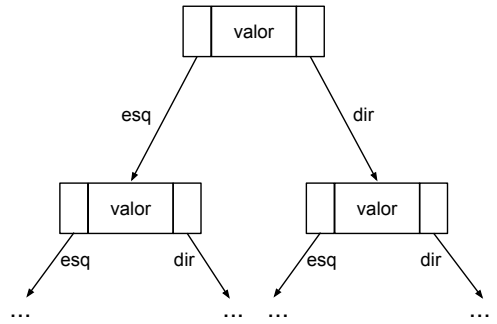

Figura 2: Ponteiros de nós em árvores

Comportamento do nó em listas

Em uma lista, o **nó** que aponta para a direita, deve apontar para o nó sucessor na lista e, o **nó** que aponta para a esquerda, deve apontar para o nó anterior na lista. Vale ressaltar que, o primeiro **nó** da lista, não aponta para nenhum outro nó à sua esquerda e, o último **nó** da lista, não aponta para nenhum outro nó à sua direita. Vide Figura 1.

Comportamento do nó em árvores

Em uma árvore, o **nó** que aponta para a direita, deve apontar para o nó filho com valor maior que o nó em questão, enquanto, o **nó** que aponta para a esquerda, deve apontar para o nó filho com valor menor que o nó em questão. Vide Figura 2.

Operações

1. Criar Coleção: Aloca a estrutura Coleção com base no tipo de estrutura especificado;

2. Criar Nó: Cria um nó com um valor especificado. A atribuição dos ponteiros do nó deve ser feita durante a inserção de valores.

3. Adicionar valor: Deve inserir o valor de acordo com o tipo de estrutura especificada na criação da Coleção. Valores duplicados são permitidos apenas nas estruturas de listas, mas não nas estruturas de árvores. Em estruturas baseadas em árvore o valor duplicado não deve alterar a árvore e nem emitir mensagens de alerta. Ao inserir um valor na estrutura do *tipo 2* é necessário percorrer **todos** os valores já incluídos na lista, para então adicionar o valor escolhido na última posição, ou seja, a implementação não deve considerar a existência de um ponteiro armazenando o último nó da lista.

4. Existe valor: Deve procurar/buscar por um valor, retornando se o valor existe ou não na estrutura.

5. Destrói estrutura: Deve desalocar todas as regiões da memória alocada pela estrutura.

Objetivo

Seu objetivo nesta primeira parte do projeto é implementar as **5 operações** básicas da estrutura **Coleção**. Para auxiliar nesta tarefa, fornecemos arquivos que **devem** ser usados como base de sua implementação. O arquivo *main.c* contém todas as chamadas que serão utilizadas na parte 2 do projeto, e portanto, não devem ser alteradas. Os arquivos *auxiliar.c* e *auxiliar.h* contém funções que não estão relacionadas à **Coleção**, mas que facilitam o desenvolvimento e também não devem ser alterados. O arquivo *Makefile* está padronizado para compilar todo o código-fonte de acordo com o esperado, e portanto não há necessidade de ser alterado. O arquivo *colecao.h* determina as definições da estrutura proposta neste projeto, e **não deve** ser alterado de qualquer forma. E por fim, o arquivo *colecao.c* é disponibilizado para que você implemente a estrutura de acordo com o esperado. É importante lembrar que funções adicionais podem ser criadas para facilitar sua implementação.

Parte 2 - Avaliação de Tempo da Estrutura

Nota: 3 pontos

Nessa segunda parte do projeto tem como objetivo analisar as Operações 3 e 4 implementadas na Parte 1 em relação aos tipos de estruturas suportados pela Coleção. Como base na sua implementação e nos resultados de tempo de inserção e busca reportados pela função *main* do projeto, você deve elaborar um relatório em PDF respondendo as seguintes perguntas:

1. No *tipo de estrutura 2*, qual seria a influência no tempo de inserção e de busca caso fosse possível utilizar um ponteiro auxiliar que apontasse para a ultima posição da lista?

2. Dos tipos de estruturas baseadas em lista, qual delas tem o maior e menor tempo de inserção? Justifique sua resposta.

3. Como os valores de entrada fornecidos para este projeto influenciam no tempo de busca em estruturas baseadas em listas? A ordem dos valores inseridos e buscados importa? O tamanho dos valores importa? A quantidade de valores importa?

4. Existe alguma diferença significativa no tempo de busca entre os tipos de estruturas baseadas em árvore? Justifique sua resposta.

5. De acordo com valores de entrada fornecidos para este projeto, a estrutura *tipo 5* melhora o tempo de busca em relação ao *tipo 4*? Justifique sua resposta.

6. De todas estruturas implementadas, qual é a mais recomendada em ambos os casos de inserção e busca para este projeto? Justifique sua resposta.

7. Na sua opinião, quais foram os desafios e aprendizados durante a implementação do projeto?