INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: BUSCA BINÁRIA E HASHTABLE

Prática 06

Parte 0: Preparação

Passo 1: Crie um novo projeto chamado **Pratica06**.

Parte 1: Implementando Lista Ordenada e Busca Binária

Passo 1: Utilize o arquivo **ordenada.cpp** que acompanha a prática 6 ao projeto.

> A função `main()` nesse arquivo realiza uma série inserções na lista, valida e exibe o seu conteúdo. Em seguida faz uma série de buscas tanto de forma sequencial como binária, exibindo o índice do elemento (caso tenha sido encontrado).

Passo 2: Implemente os métodos que faltam na classe `ListaOrdenada`, seguindo as orientações abaixo:

> Método `insere()`: insere o elemento na lista de forma a mantê-la ordenada. Dica de implementação: varra a lista de trás para frete, movendo para a direita os elementos maiores que o elemento a ser inserido. Quando achar um item da lista menor ou igual ao elemento, parar e inserir na posição seguinte.
>
> Método `remove()`: remove o elemento da lista mantendo a ordenação. Dica: use a busca binária para achar a posição do elemento (se existir), depois movimente os itens à direita dele uma posição para a esquerda.
>
> Método `buscaSequencial()`: busca o elemento na lista sequencialmente a partir do começo, retornando o índice quando é encontrado ou -1 caso contrário (isto é, chegue ao final sem encontrar). Para a busca ser mais otimizada, tire vantagem da ordenação: no momento que for encontrando um elemento maior que a chave que buscamos, podemos encerrar a busca sem sucesso (-1).
>
> Método `buscaBinaria()` [privado]: esse método realiza a busca do elemento na lista empregando a técnica de busca binária descrita no material de aula. (A implementação pode ser recursiva ou iterativa, a seu critério).

Passo 3: Rode e teste a aplicação.

> Verifique se a lista está válida e se o resultado das buscas está correto. Pode haver divergência no resultado das buscas quando há repetição de valores na lista.

Passo 4: **(Desafio/Opcional)** Faça testes comparando o tempo de busca usando a busca sequencial e a busca binária.

> Inicialize a lista com milhares de elementos e faça uma série de buscas de cada tipo (também na ordem de milhares) para ver o tempo que cada um toma.

Parte 2: Implementando Tabela de Espalhamento (Hashtable)

Passo 1: Adicione ao projeto o arquivo **hashtable.cpp** da prática e estude o código existente.

Nesse arquivo está implementada uma tabela de espalhamento que trabalha com entradas na forma (chave, valor) ou (*key*, *value*). No construtor dessa classe é indicado o tamanho da tabela.

Do ponto de vista de implementação, a tabela usa um *array* de listas ligadas (isto é, com ponteiros) chamado *table* para armazenar os pares (chave, valor). Por exemplo, ao inserir na tabela, busca-se primeiro a posição correta no *array* (através da função de *hash*), a qual aponta para um lista encadeada contendo os elementos. A lista é usada para resolver as colisões (várias chaves mapeando para a mesma posição do *array*). Estude a classe `List` nesse código para entender como deve ser usada.

A função `main()` instancia duas tabelas de espalhamento: uma de (`int`, `string`) para simular (matrículas, alunos); e outra de (`string`, `float`), para simular (aluno, nota). São feitas inserções e buscas nessas tabelas.

Passo 2: Implemente os métodos que faltam na classe `Hashtable`, seguindo as orientações abaixo:

Método `insert(key, value)`: esse método deve usar a função `hash()` privada para gerar um número inteiro a partir da chave (`key`) que será o índice na tabela (considerando a capacidade, usar resto da divisão). Esse índice aponta para lista encadeada onde o item deve ser adicionado (`add()`).

Método `remove(key)`: semelhante ao `insert(key, value)`, primeiro é gerado um índice para a tabela, que é usado para chamar o método `remove()` para a lista correspondente.

Método `search(key, notFound)`: como nos outros, primeiro obtêm-se o hash/índice da chave, o qual indica lista. Deve ser feita uma busca pela chave nessa lista usando `search()` e retornar o valor encontrado. Em caso de exceção `ItemNotFound`, retornar o valor `notFound`.

Método(s) privado(s) `hash(key)`: esses métodos devem calcular um índice a partir da chave. Boas funções de espalhamento devem fazer com que os itens fiquem o mais espalhados o possível na tabela, isto é, evitar colisões. Serão necessários dois métodos para a prática: um para chaves inteiras e outro para chaves do tipo string.

Passo 3: Rode e teste a aplicação.

Verifique se a tabela é válida e se o resultado das buscas está correto.