Escola de Artes, Ciências e Humanidades (USP-Leste)
Algoritmos e Estruturas de Dados I - 2º Semestre de 2015
Prof. Alexandre da Silva Freire (afreire@ime.usp.br – sala 322K-I1)

Aula 2 - 07/08 - Listas Lineares - Deques - Listas Duplamente Ligadas

Listas Lineares

Uma *lista linear* é uma coleção de elementos organizados em alguma ordem total arbitrária (não há restrições que a ordem deve satisfazer). A noção de ordem total serve apenas para nos permitir criar operações para listar os elementos na ordem em que eles estão organizados, inserir novos elementos em qualquer posição da lista, remover elementos mantendo a ordem relativa dos elementos restantes, etc... Em nossa disciplina, assumiremos que os elementos são sempre números inteiros. Na prática, os elementos podem ser qualquer tipo de objeto (exemplos: textos, fotos, videos, etc...) e cada objeto está sempre associado a uma *chave* numérica (exemplo: a ficha que contém os dados cadastrais de um aluno da USP pode estar associada ao número USP do respectivo aluno). De forma geral, em uma lista linear não há restrições sobre como os elementos são inseridos ou removidos. Ou seja, pode-se inserir um novo elemento em qualquer posição da lista, assim como pode-se remover qualquer elemento da lista.

Deques

Os *deques*[1] são listas lineares com a restrição de que só pode haver inserção e remoção nas pontas da lista. Ou seja, insere-se um novo elemento sempre na primeira ou na última posição e um elemento só pode ser removido da lista se este for o último ou o primeiro da lista. Há outras estruturas de dados que são listas lineares ainda mais restritas, sendo as *pilhas* e as *filas* as mais utilizadas e, por isso, as que nos interessam nesse curso. Após compreender como funcionam os deques, fica fácil deduzir como funcionam as filas e as pilhas. Por isso, iremos direto ao ponto!

Listas Duplamente Ligadas

A primeira maneira que veremos de se implementar um *deque* é utilizando *listas duplamente ligadas* (LDL). Em uma LDL, utilizamos uma estrutura de nó, que contém um elemento e aponta para os nós vizinhos na lista (próximo e anterior), conforme o código java abaixo.

[1] O termo *deque* é uma abreviação de *double-ended queue*.

```
class No {
   int elemento;
   No proximo;
   No anterior;
   No(int elemento) {
      this.elemento = elemento;
   }
}
```

Os campos `proximo` do último nó e `anterior` do primeiro nó devem ser iguais a `null`. A classe que representa a lista em si deve manter referências para os elementos das pontas (ou seja, devem conter os campos `No primeiro` e `No ultimo`) e deve implementar a interface `Deque`, especificada abaixo. Quando a lista estiver vazia (sem nenhum elemento), os campos `primeiro` e `ultimo` devem ser iguais a `null`.

```
interface Deque {
   boolean vazio();
   void insereNoInicio(int elemento);
   int primeiroElemento();
   int removePrimeiroElemento();
   void insereNoFinal(int elemento);
   int ultimoElemento();
   int removeUltimoElemento();
   Iterador iterador();
}
```

Pelos nomes dos métodos da interface `Deque` fica claro o que cada método deve fazer, exceto talvez o último método, que devolve um iterador do deque. Como toda interface, a interface `Deque` abstrai seus métodos de suas respectivas implementações. Para que o usuário dessa interface consiga varrer o deque, sem necessariamente ter conhecimento de como ele está implementado, tem-se a necessidade de que uma classes implemente a interface `Iterador` descrita abaixo.

```
interface Iterador {
   boolean temProximo();
   int proximo();
}
```

A classe que implementar a interface `Iterador` deve conhecer os detalhes

de implementação do deque em questão. Ou seja, cada implementação do deque deve prover também sua respectiva implementação de um iterador.

Para ilustrar como um iterador é tipicamente utilizado, considere o seguinte trecho de código que imprime todos os elementos da lista, na ordem em que eles estão organizados:

```
for (Iterador it = deque.iterador(); it.temProximo();) {
    System.out.print(it.proximo());
}
```

Exercício

Entre no Tidia e baixe o arquivo `aula02.zip` da pasta `códigos`. Implemente a classe `DequeListaDuplamenteLigada.java` conforme discutido em aula.

Dica 1: durante a escrita de seu código, faça desenhos para ilustrar cada passo que seus métodos estão executando.

Dica 2: embora em java não seja possível liberar diretamente a memória, como na linguagem C, temos que nos preocupar em deixar "inalcansáveis" os objetos que não serão mais utilizados, para que o *garbage colector* libere a memória de forma adequada. Se você ainda não sabe como funciona a "coleta de lixo" do java, recomendo fortemente que pesquise sobre o tema.