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 4 - 14/08 - Comentários sobre os exercícios da aula passada - Filas - Pilhas

Comentários sobre os exercícios da aula passada

O primeiro exercício da aula passada pedia para implementar a interface `Deque` através de uma lista ligada simples (na qual a classe `No` não possui um apontador `anterior`). No geral, o código da lista ligada (simples) é basicamente o mesmo da lista duplamente ligada, apenas algumas linhas são removidas e a eficiência dos métodos permanece a mesma. A única exceção é o método `removeUltimoElemento`, que agora tem seu consumo de tempo dependendo do tamanho da lista, o que na lista duplamente ligada não acontecia. Ao remover o último nó, o nó que assumirá a "ponta direita" (o novo último nó) da lista será o penúltimo nó. Como não se possui um apontador para o nó anterior ao último, há a necessidade de percorrer a lista desde o início até chegar ao penúltimo nó e, portanto, essa operação tende a ficar mais lenta conforme o tamanho da lista aumenta.

O segundo exercício da aula passada pedia para implementar a interface `Deque` através de um vetor redimensionável. A seguir, propomos duas estratégias diferentes para gerenciar o tamanho do vetor. No momento, não dispomos de ferramentas matemáticas suficientes para comparar a eficiência dessas duas estratégia. Por equanto, vamos nos deter apenas a apresentar as estratégias, sem compará-las.

Estratégia de aumento constante

A primeira estratégia consiste em aumentar e diminuir o vetor sempre em um tamanho constante. No código, essa constante é representada pela variável `TAMANHO_AUMENTO`). Abaixo, mostramos o código dos métodos responsáveis por aumentar e diminuir o tamanho do vetor.

```
void aumentaTamanho() {
   redimenciona(vetor.length + TAMANHO_AUMENTO);
}

void diminuiTamanho() {
   redimenciona(vetor.length - TAMANHO_AUMENTO);
}
```

Uma vez definida a estratégia de como aumentaremos e diminuiremos o tamanho do vetor, precisamos estabelecer em quais situações esses redi-

mensionamentos são realizados. Nossa proposta é de que só aumentemos o tamanho do vetor quando não há mais espaço para uma possível inserção no início ou no final. Propomos que o vetor tenha seu tamanho diminuído quando após realizar tal operação o vetor esteja com espaço livre equivalente a como se tivéssemos realizado um aumento sobre um vetor "cheio". Abaixo, apresentamos esses códigos:

```
boolean precisaAumentar() {
   return Math.abs(ultimo - primeiro + 1) >= vetor.length - 1;
}

boolean precisaDiminuir() {
   return Math.abs(ultimo - primeiro + 1) <= vetor.length
             - (2 * TAMANHO_AUMENTO);
}
```

Estratégia de dobrar o tamanho

A segunda estratégia que propomos é a de dobrar o tamanho do vetor quando o mesmo estiver "cheio" e, quando o vetor estiver "vazio", diminuir pela metada o seu tamanho. Assim como na estratégia anterior, diminui-se o tamanho do vetor quando após realizar tal operação o vetor esteja com espaço livre equivalente a como se tivéssemos realizado um aumento sobre um vetor "cheio". Repare que, como o aumento agora é feito dobrando-se o tamanho do vetor, a diminuição deve ocorrer quando o vetor estiver com apenas 1/4 de seu espaço preenchido. Abaixo, apresentamos esses códigos:

```
void aumentaTamanho() {
   redimenciona(vetor.length * 2);
}

void diminuiTamanho() {
   redimenciona(vetor.length / 2);
}

boolean precisaAumentar() {
   return Math.abs(ultimo - primeiro + 1) >= vetor.length - 1;
}

boolean precisaDiminuir() {
   return Math.abs(ultimo - primeiro + 1) <= vetor.length / 4;
}
```

Filas e Pilhas

As *pilhas* e as *filas* são listas lineares ainda mais restritas que os deques. Essas estruturas se comportam exatamente conforme a noção intuitiva sugerida pelos seus nomes. Ou seja, numa fila, o primeiro elemento a entrar é

o primeiro a sair. E numa pilha, o último elemento a entrar é o primeiro a sair. Em outras palavras, numa fila insere-se apenas no final da lista (final da fila) e remove-se apenas o primeiro elemento da lista (início da fila). E numa pilha, as inserções e remoções são feitas apenas no final da lista (no topo da pilha).

Exercícios

Entre no Tidia e baixe o arquivo `aula04.zip` da pasta `códigos`.

Exercício 3. Implemente as interfaces `Fila` e `Pilha` utilizando uma lista ligada (simples). As inserções e remoções devem ser feitas da forma mais eficiente possível.