CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DO RN

GERÊNCIA DE TECNOLOGIA DA INFORMAÇÃO E EDUCACIONAL DE TELEMÁTICA

Exercícios de Estrutura de Dados I

1. Implemente uma classe `VetorArray`, que implemente a interface `Vetor` (baixe da página da disciplina). Faça uma tabela com os tempos de execução de cada método. Use a implementação da classe `java.util.Vector` de JAVA como referência, também disponível na página.

2. Descreva em pseudo-código e depois implemente em JAVA os algoritmos para os métodos a seguir de um TAD lista implementado com uma lista duplamente encadeada (baixe a interface *List.java* e uma implementação *DLLList.java* – Precisa da interface *Position*):

   (a) Position *last()*

   (b) Position *next(p)*

   (c) Position *insertAfter(p , element)*

   (d) Position *insertLast(element)*

3. Considere o seguinte algoritmo para o método *insertAfter(p,e)*:

```
public Position insertAfter(p, e){
  DNode no1 = checkPosition(p);
  DNode no2 = new DNode();
  no2.setElement(e);
  no2.setPrev(no1);
  no2.setNext(no1.getNext());
  no1.setNext(no2);
  (no1.getNext()).setPrev(no2);
  return no2;
}
```

   O método está certo? Justifique.

   Dica: Desenhe as referências após cada linha do algoritmo

4. Faça uma implementação de uma Lista usando arranjos (*array*) e outra usando Lista Ligada simples. Faça uma tabela com os tempos de execução de todos os métodos de *List* para as três implementações (Lista ligada simples, Lista duplamente ligada e arranjos).

5. Desenvolva um método **recursivo** `public Position find(int i)` que retorne a posição de um inteiro i em uma lista de inteiros. O método deverá disparar a exceção `ValueNotFoundException`.

6. Desenhe uma figura que representa os elementos de uma sequência S (inicialmente vazia) depois das seguintes operações:

   (a) S.insertLast(a)

   (b) S.insertFirst(b)

   (c) S.insertAtRank(1,c)

   (d) S.remove(S.after(S.first()))

   (e) S.insertAfter(S.last(),d)

7. Considere uma sequência implementada com uma lista duplamente encadeada (os nós são da classe NoSeq) e que possui nós sentinelas que indicam os primeiro e o último elementos. Preencha a linha em branco de uma implementação do método `insertBefore` em JAVA. Assuma que os nós implementam a interface `Position`.

```
public Position insertBefore(Position p, Object e)
throws InvalidPositionException{
    NoSeq temp = new NoSeq();
    temp.setElement(e);
    temp.setNext(p);
    temp.setPrev(p.getPrev());

    ---------------------------------
    p.setPrev(temp);
    return temp;
}
```

8. Implemente uma sequência usando o objeto `Vector` do JAVA.

9. Implemente uma classe `VetorLista`, que implemente a interface *Vetor*, e que use como representação interna uma lista duplamente ligada. Faça tabela comparando os tempos de execução das duas implementações do Vetor (*array* e Lista Ligada).

10. Faça uma tabela com os tempos de execução dos métodos de Sequência implementados com arranjo e com Lista duplamente ligada. Descreva os algoritmo dos métodos que possuem tempos de execução diferente nas duas implementações e explique porque os tempo são diferentes.

11. Um programa P usa uma implementação de sequência como seu componente principal. Sabe-se que P usa as operações **atRank**, **insertAtRank** e **remove** em ordem não determinada. Sabe-se também que P realiza n^2 operações **atRank**, $2n$ operações **insertAtRank** e n operações **remove**. Qual implementação do TAD sequência deve ser usada para obter maior eficiência: a implementação baseada em *array* ou a implementação baseada em lista duplamente encadeada? Justifique sua resposta.

12. Considere um novo método no TAD **Sequencia** (implementado com lista duplamente ligada), chamado *makeFirst(p)*, que move o elemento na posição p de uma sequência S para a primeira posição de S. O método deve manter a ordem relativa de todos os outros elementos e deve executar em tempo $O(1)$.

13. Adicione o método *shrinkToFit()* à sua classe `VetorArray`, que substitua o *array* atual por um *array* exatamente do tamanho do Vetor.

14. Faça um programa em JAVA para testar as classes que implementem a interface *List*. Imagine uma lista de itens, onde você pode acrescentar e remover itens no fim ou no início da lista. Além disso, você tem um "cursor" que indica o item em evidência, e você pode inserir itens antes ou depois deste item, além de poder removê-lo. Também considere botões de navegação, nos quais você pode mudar o item em evidencia, também pode ir para o primeiro ou o último item.

    Este é o modelo aproximado usado em sistemas on-line de comprar, onde você tem o carrinho de compras e pode manipular os itens do carrinho.

    Dica: Você pode fazer um servlet *que liste todos os itens da lista e deixe o item em evidência com uma cor diferente. No fim da página coloque botões que servem para manipular a lista (Inserir Início, Inserir Fim, Remover Atual, Próximo, Anterior, Inserir Depois, Inserir Antes, Primeiro, Último, Quantidade de Itens) e uma caixa de texto para digitar uma String a ser usada quando for inserir um novo item.*

15. Problema da reordenação:

Dada uma sequência de n elementos, podemos querer que estes elementos mantenham uma relação de ordem. Consideramos que quaisquer dois elementos consecutivos possam ser comparados para determinar se estão na ordem desejadas e, se não estiverem, eles podem ser trocados de posição.

O **algoritmo da bolha** usa esse princípio para reordenar os elementos. A idéia é fazer uma série de passagens trocando de colocações/posições elementos que não estão na ordem desejada. Em cada passagem, os elementos são **varridos** incrementando-se a colocação (de 0 até $n-1$). A sequência estará ordenada após n passagens.

O algoritmo tem os seguintes princípios:

(a) Na primeira pasagem, o maior elemento será **empurrado** até a última colocação, pois sua colocação será trocada sempre que o seu sucessor for menor do que ele.

(b) Na segunda passagem, o segundo maior será empurrado até a penúltima colocação, uma vez que na última comparação o seu sucessor será maior do que ele (observe que está última comparação pode não ser feita).

(c) Assim será repetido até o primeiro elemento.

Ao final da i-ésima passagem os i elementos mais a direita (da colocação $n-i$ até a colocação $n-1$) estarão em suas colocações finais (colocações desejadas na ordem estabelecida). Dada estas informações, podemos limitar o número de passagens em uma sequência de n elementos em n. Além disso podemos limitar que na i-ésima passagem apenas os primeiros $n-i-1$ elementos sejam varridos.

O método a seguir (em JAVA) implementa o método da bolha:

```
public static void bolha1(Sequencia S)
  int n = S.size();
  for (int i = 0 ; i < n ; i++ )
    for (int j = 1 ; j < n-i ; j++ )
      if (S.elemAtRank(j-1) > S.elemAtRank(j))
        S.swapElements(S.atRank(j-1), S.atRank(j))
}
```

Responda:

(a) Qual a ordem de complexidade do algoritmo e porque?

  i. Se o TAD sequência for implementado usando um *array*.

  ii. Se o TAD sequência for implementado usando uma lista duplamente encadeada.

(b) Desenvolva uma algoritmo `public static void bolha2(Sequencia S)` que implemente o método da bolha usando apenas posições da Sequência

(c) Qual a ordem de complexidade do algoritmo que você desenvolveu e por que?

  i. Se o TAD sequência for implementado usando um *array*.

  ii. Se o TAD sequência for implementado usando uma lista duplamente encadeada.

Sites sobre método da bolha:

- `http://www.cs.brockport.edu/cs/java/apps/sorters/bubblesort.html`
- `http://www.cs.princeton.edu/~ah/alg_anim/gawain-4.0/BubbleSort.html`
- `http://math.hws.edu/TMCM/java/xSortLab/`
- `http://www.cs.rit.edu/~atk/Java/Sorting/sorting.html`

Dica para a prática: faça um programa que leia números inteiros de um arquivo e escreva um outro arquivo com os números ordenados. Os nomes dos arquivos podem ser parâmetros passados para o método *main*. Primeira leia todos os números colocando-os em uma sequência, ordene a sequência e depois escreva em um arquivo.