

Listas encadeadas

Embora os algoritmos de manipulação de listas lineares com alocação estática sejam relativamente simples, seu uso prático é limitado. Particularmente se for considerado a otimização de uso da memória. Duas situações são bastante comuns: ou o tamanho da lista é superdimensionado para que o programador tenha certeza que ela terá espaço suficiente para armazenar todos os seus dados, havendo neste caso um possível desperdício de memória, ou a lista é dimensionada com o tamanho inferior ao necessário – nesta situação ou tempo de processamento adicional deve ser inserido para criar uma nova lista, maior que a original, e copiar os dados de uma lista para outra. A solução é utilizar listas alocadas dinamicamente.

Em listas dinâmicas, cada novo dado inserido na lista é encapsulado em um nó. Este nó é alocado dinamicamente, ou seja, conforme necessidade do programa em execução e inserido na lista. Conforme os dados são consumidos da lista, os nós podem ser descartados. Note que uma estratégia dinâmica de manipulação de lista consome um tempo maior de processamento, pois requer alocação e liberação dinâmica de memória, no entanto, na prática, o ganho e a versatilidade frente a estratégias estáticas é muito grande.

Importante observar que todas as políticas já vistas para listas continuam sendo válidas. No entanto, particular interesse existe quando não é possível prever a quantidade de dados a ser manipulada ou quando se faz necessário algum tipo de ordenação das informações armazenadas

Noção de nó

Em uma lista encadeada, o principal elemento é denominado *nó* ou *nodo*. Um nó encontra-se em uma determinada posição da lista, sendo a lista uma sucessão de nós. Cada nó contém, no mínimo, dois campos: uma refere-se ao dado armazenado na lista naquela posição, a outra refere-se a um ponteiro a outro nó na mesma lista. O dado é a própria informação da aplicação, o ponteiro (ou ponteiros, pois podem existir dois ou mais ponteiros) permite o encadeamento da lista. Denomina-se o primeiro nó da lista de cabeça e o último de último.

Lista simplesmente encadeada

Uma lista simplesmente encadeada é uma sucessão de nós onde cada nó aponta para o próximo nó da lista. O nó que possuir o valor `null` no ponteiro para próximo é o último nó da lista. É de extrema importância que seja mantida uma referência para o primeiro nó da lista, caso esta referência for `null`, significa que a lista esta vazia. Em certas situações também é útil possuir uma referência ao último nó.

Abaixo uma estrutura de código para dois tipos de lista: uma Fila e uma Pilha. Note que uma estrutura genérica Lista foi concebida.

1.	<code>import java.io.*;</code>
2.	
3.	<code>abstract class Lista {</code>
4.	<code> protected class Elem {</code>
5.	<code> private Object ob;</code>
6.	<code> private Elem next;</code>
7.	
8.	<code> public Elem(Object ob, Elem next) {</code>
9.	<code> this.ob = ob;</code>
10.	<code> this.next = next;</code>
11.	<code> }</code>
12.	<code> public Object getElem() {</code>
13.	<code> return ob;</code>
14.	<code> }</code>
15.	<code> public Elem getNext() {</code>
16.	<code> return next;</code>
17.	<code> }</code>
18.	<code> public void setNext(Elem proximo) {</code>
19.	<code> next = proximo;</code>
20.	<code> }</code>
21.	<code> } //Elem</code>
22.	<code> protected Elem prim;</code>
23.	<code> protected int nbelem;</code>
24.	
25.	<code> public Lista() {</code>
26.	<code> nbelem = 0;</code>
27.	<code> prim = null;</code>
28.	<code> }</code>
29.	<code> public int length() {</code>
30.	<code> return nbelem;</code>
31.	<code> }</code>
32.	<code> abstract void insere(Object ob);</code>
33.	<code> abstract Object retira();</code>

34.	}
35.	
36.	class Fila extends Lista {
37.	public Fila() {
38.	super();
39.	}
40.	public void insere(Object ob) {
41.	nbelem++;
42.	Elem aux = new Elem(ob, prim);
43.	prim = aux;
44.	}
45.	public Object retira() {
46.	Object aux = null;
47.	
48.	if(nbelem == 1) {
49.	nbelem = 0;
50.	aux = prim.getElem();
51.	prim = null;
52.	}
53.	else if(nbelem > 1) {
54.	Elem penultimo = prim,
55.	ultimo = prim.getNext();
56.	for(; ultimo.getNext() != null ; ultimo = ultimo.getNext())
57.	penultimo = ultimo;
58.	aux = ultimo.getElem();
59.	penultimo.setNext(null);
60.	nbelem--;
61.	}
62.	return aux;
63.	}
64.	}
65.	
66.	class Pilha extends Lista {
67.	public Pilha() {
68.	super();
69.	}
70.	public void insere(Object ob) {
71.	nbelem++;
72.	Elem aux = prim;
73.	prim = new Elem(ob, aux);
74.	}
75.	public Object retira() {
76.	Object aux = null;
77.	
78.	if(nbelem > 0) {
79.	aux = prim.getElem();
80.	nbelem--;
81.	prim = prim.getNext();
82.	}
83.	return aux;
84.	}
85.	public void push(Object ob) {
86.	this.insere(ob);
87.	}
88.	public Object pop() {
89.	return this.retira();
90.	}
91.	}
92.	
93.	class ManipulaLista {
94.	private Lista l;
95.	
96.	public ManipulaLista() {
97.	l = new Fila();
98.	}
99.	public ManipulaLista(Lista l) {
100.	this.l = l;
101.	}
102.	public void encheLista() {
103.	String linha;

104.	<code>InputStreamReader teclado = new InputStreamReader(System.in);</code>
105.	<code>BufferedReader console = new BufferedReader(teclado);</code>
106.	
107.	<code>for(int i = 0 ; i < 10 ; i++) {</code>
108.	<code> try {</code>
109.	<code> System.out.print("Entre texto " + (i+1) + " de 10: ");</code>
110.	<code> linha = console.readLine();</code>
111.	<code> }</code>
112.	<code> catch (Exception e) {</code>
113.	<code> System.out.println("Nao conseguiu ler...");</code>
114.	<code> return;</code>
115.	<code> }</code>
116.	<code> //l.insere(new String(linha));</code>
117.	<code> l.insere(linha);</code>
118.	<code> }</code>
119.	<code>}</code>
120.	<code>public void esvaziaLista() {</code>
121.	<code> for(int i = l.length() ; i > 0 ; i--)</code>
122.	<code> System.out.println("Retirado: " + l.retira());</code>
123.	<code> }</code>
124.	<code>}</code>
125.	
126.	<code>public class UsaLista {</code>
127.	<code> public static void main (String args[]) {</code>
128.	<code> ManipulaLista ml;</code>
129.	
130.	<code> System.out.println("Manipulando uma Fila");</code>
131.	<code> Fila f = new Fila();</code>
132.	<code> ml = new ManipulaLista(f);</code>
133.	<code> ml.encheLista();</code>
134.	<code> ml.esvaziaLista();</code>
135.	
136.	<code> System.out.println("Manipulando uma Pilha");</code>
137.	<code> Pilha p = new Pilha();</code>
138.	<code> ml= new ManipulaLista(p);</code>
139.	<code> ml.encheLista();</code>
140.	<code> ml.esvaziaLista();</code>
141.	<code> }</code>
142.	<code>}</code>

Exemplo de uso: compilação e execução. Ambiente GNU-Linux, JDK 1.4.2.10

```
gersonc@anahy ~/fnt/Java $ javac UsaLista.java
gersonc@anahy ~/fnt/Java $ java UsaLista
Manipulando uma Fila
Entre texto 1 de 10: um
Entre texto 2 de 10: dois
Entre texto 3 de 10: tres
Entre texto 4 de 10: quatro
Entre texto 5 de 10: cinco
Entre texto 6 de 10: seis
Entre texto 7 de 10: sete
Entre texto 8 de 10: oito
Entre texto 9 de 10: nove
Entre texto 10 de 10: dez
Retirado: um
Retirado: dois
Retirado: tres
Retirado: quatro
Retirado: cinco
Retirado: seis
Retirado: sete
Retirado: oito
Retirado: nove
Retirado: dez
Manipulando uma Pilha
Entre texto 1 de 10: um
Entre texto 2 de 10: dois
Entre texto 3 de 10: tres
Entre texto 4 de 10: quatro
Entre texto 5 de 10: cinco
```

```
Entre texto 6 de 10: seis
Entre texto 7 de 10: sete
Entre texto 8 de 10: oito
Entre texto 9 de 10: nove
Entre texto 10 de 10: dez
Retirado: dez
Retirado: nove
Retirado: oito
Retirado: sete
Retirado: seis
Retirado: cinco
Retirado: quatro
Retirado: tres
Retirado: dois
Retirado: um
gersonc@anahy ~/fnt/Java $
```

Exercícios:

1. O programa neste material apresenta um exemplo prático da aplicação de diversos conceitos da programação orientada a objetos na linguagem Java. Em adição, diferentes aspectos próprios à linguagem Java (como tratamento de exceções, manipulação de *streams* e classe `Object`) são explorados. Como exercício, o aluno deve:
 - Comentar, linha a linha, do código fonte, identificando o uso dos conceitos do paradigma de programação orientado a objeto.
 - Recuperar o arquivo fonte, compilá-lo, e executá-lo passo a passo, compreendendo tanto o algoritmo implementado como o emprego dos conceitos da orientação a objetos. Nesta atividade, o aluno deve reproduzir a representação da memória.
2. Crie uma classe `FilaDeInteirosComPrioridade`, onde a inserção de um novo valor inteiro é realizada respeitando a ordem aritmética.
3. Modifique a classe criada no exercício anterior de tal forma que seja possível manter ordenado objetos quaisquer (como `String`, `Cofrinho`, ...) . Consulte em <http://java.sun.com/j2se/1.5.0/docs/api/> a classe `Comparable`, em particular o método virtual `compareTo`.
4. Modifique a classe `Fila` do programa exemplo com um método capaz de retirar um determinado elemento da fila.
5. Modifique a classe `Fila` do programa exemplo com um método capaz de ordenar os elementos na fila.
6. Analise o código do método `remove` da classe `Fila`. Como ele funciona? Como ele poderia ser melhorado?

Lista duplamente encadeada

Uma lista duplamente encadeada é uma sucessão de nós onde cada nó aponta para o próximo nó da lista e para seu predecessor. Assim, além do campo relativo ao dado, cada nó possui dois ponteiros, que chamaremos de `prox` e `ant`. O objetivo do duplo encadeamento é tornar mais simples e mais eficiente a execução dos algoritmos.

Exercícios:

7. Reimplemente a classe `Fila`, utilizando agora o mecanismo de duplo encadeamento.
8. Refaça os exercícios 4 e 5 considerando a implementação do Exercício 7.
9. Crie uma classe `DeQueue`, que possibilita a inserção e remoção de elementos tanto no início quanto no fim da fila.