Listas Encadeadas

Fabrício J. Barth

BandTec - Faculdade de Tecnologia Bandeirantes

Fevereiro de 2011

Tópicos Principais

- Motivação
- Listas encadeadas
- Implementações recursivas
- Listas de tipos estruturados

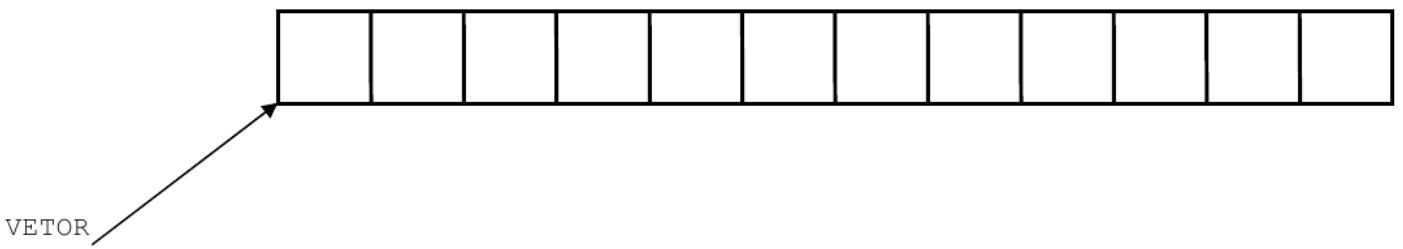
Tópicos complementares

- Listas circulares
- Listas duplamente encadeadas

Tópicos Principais

Motivação

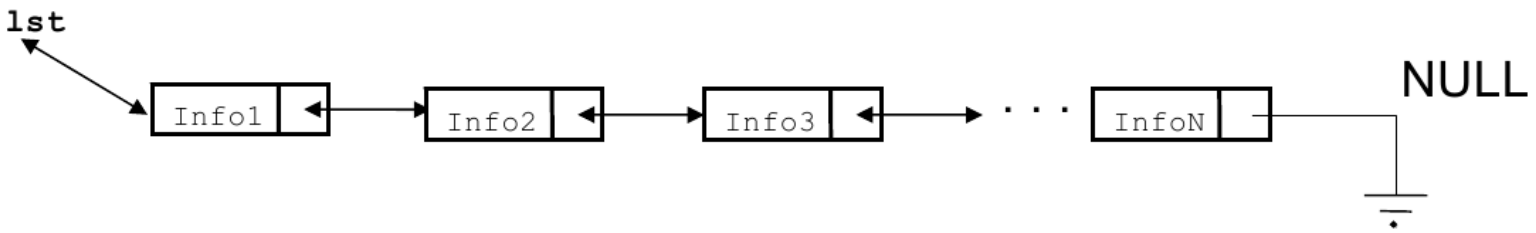
- Vetor:
  - ⋆ ocupa um espaço contíguo de memória
  - ⋆ permite acesso randômico aos elementos
  - ⋆ deve ser dimensionado com um número máximo de elementos



Motivação

- Estruturas de dados dinâmicas: crescem ou decrescem à medida que elementos são inseridos ou removidos.
- Exemplo: **listas encadeadas**.
- Listas encadeadas são amplamentes utilizadas para implementar outras estruturas de dados.

Listas Encadeadas



- sequência encadeada de elementos, chamados **nós** da lista.
- **nó da lista** é representado por dois campos:
  - ⋆ a informação armazenada e
  - ⋆ o ponteiro para o próximo elemento da lista

- a lista é representada por um ponteiro para o primeiro nó
- o ponteiro do último elemento é NULL.

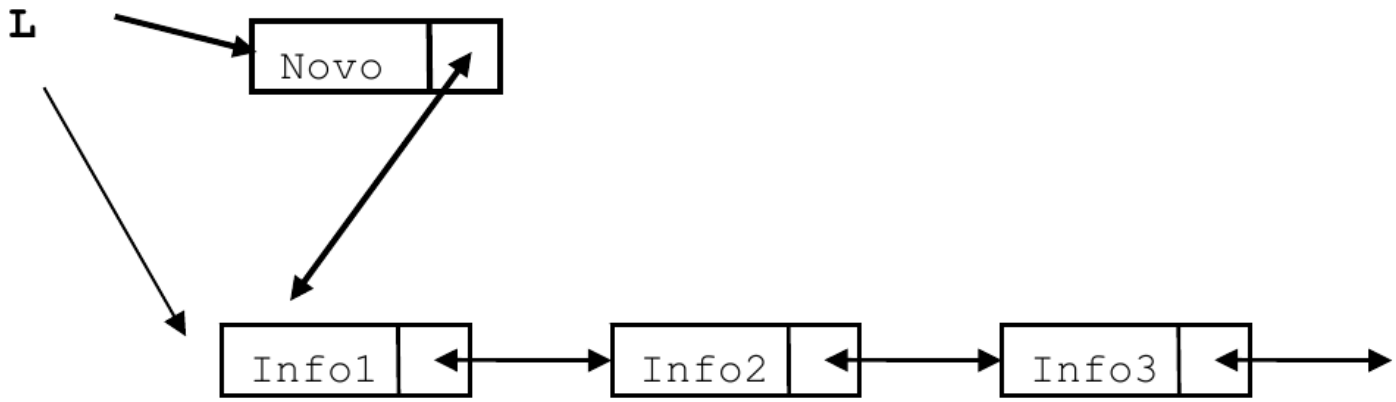
Estrutura com ponteiro para ela mesma

```
1   class Nodo {
2       private int info;
3       private Nodo prox;
4
5       public int getInfo() {
6           return info;
7       }
8       public void setInfo(int info) {
9           this.info = info;
10      }
11      public Nodo getProx() {
12          return prox;
13      }
14      public void setProx(Nodo prox) {
15          this.prox = prox;
16      }
17  }
```

Criação da lista vazia

```
1  public class Lista {
2          private Nodo prim;
3          public void criaLista(){
4                  prim = null;
5          }
6  }
```

Listas encadeadas de inteiros: inserção



- Aloca memória para armazenar o elemento
- Encadeia o elemento na lista existente

Listas encadeadas de inteiros: inserção

```
1  /*
2   * insercao no inicio
3   */
4  public void add(int i){
5      Nodo novo = new Nodo();
6      novo.setInfo(i);
7      novo.setProx(prim);
8      prim = novo;
9  }
```

Exemplo de utilização

```
1  public Main(){
2      Lista lista = new Lista();
3      lista.criaLista();
4      lista.add(45);
5      lista.add(60);
6      lista.add(1);
7  }
```

Função que percorre os elementos da lista

```
1  public void print(){
2      for(Nodo n = prim; n != null; n = n.getProx()){
3          System.out.println(n.getInfo());
4      }
5  }
```

Exemplo de utilização

```
1  public Main(){
2      Lista lista = new Lista();
3      lista.criaLista();
4      System.out.println("Imprimindo valores");
5      lista.print();
6      lista.add(45);
7      lista.add(60);
8      lista.add(1);
9      System.out.println("Imprimindo valores");
10     lista.print();
11 }
```

Função que verifica se a lista está vazia

```
1  public boolean isEmpty(){
2      if(prim == null)
3          return true;
4      else
5          return false;
6  }
```

Função de busca

- Recebe a informação referente ao elemento a pesquisar
- Retornar o objeto da lista que representa o elemento ou null, caso o elemento não seja encontrado na lista.

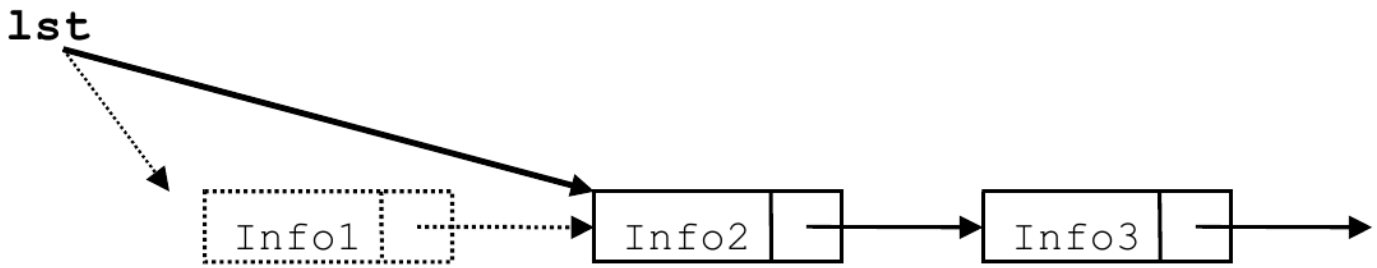
```
1  /*
2   * busca por um elemento na lista
3   */
4  public Nodo search(int i){
5      for(Nodo n = prim; n != null; n = n.getProx()){
6          if(n.getInfo()==i){
7              return n;
8          }
9      }
10     return null; /* nao achou o elemento*/
11 }
```

Exemplo de utilização da função de busca

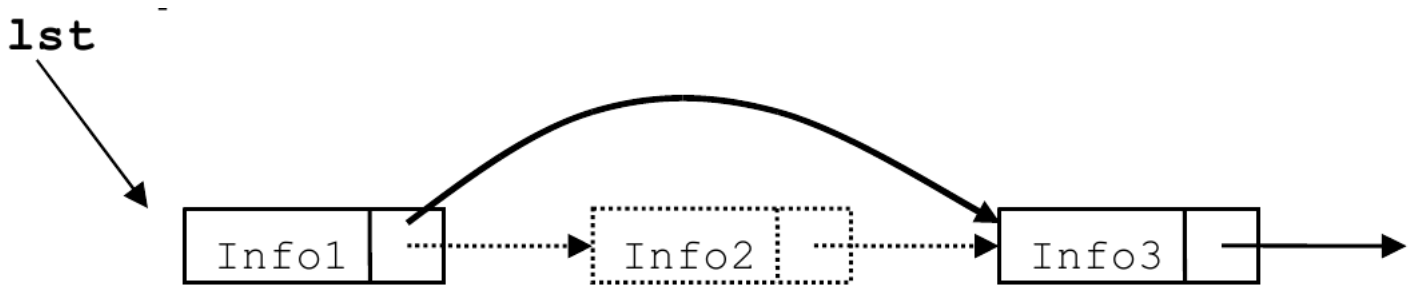
```
1  Nodo temp;
2  if((temp = lista.search(60)) != null)
3      System.out.println("Achou "+temp.getInfo());
4  else
5      System.out.println("Nao achou o elemento");
```

Função que retira um elemento da lista

- Recebe como entrada o valor do elemento a retirar.
- Atualiza o valor do ponteiro para a lista ($prim$) se o elemento removido for o primeiro.



- caso contrário, remove apenas o elemento da lista.



```
 1    public void remove(int i){
 2        /*objeto para o elemento anterior*/
 3        Nodo anterior = null;
 4        /*objeto para percorrer a lista*/
 5        Nodo p = prim;
 6
 7        /*procura elemento na lista, guardando anterior*/
 8        while(p != null && p.getInfo() != i){
 9            anterior = p;
10            p = p.getProx();
11        }
12
13        /*verifica se achou elemento*/
14        if(p == null){
15            /*nao achou: mantem prim da forma como estah*/
16            return;
17        }
18
19        /*retira elemento*/
20        if(anterior == null){
21            /*retira elemento do inicio*/
22            prim = p.getProx();
23        }else{
24            /*retira elemento do meio da lista*/
25            anterior.setProx(p.getProx());
26        }
27    }
```

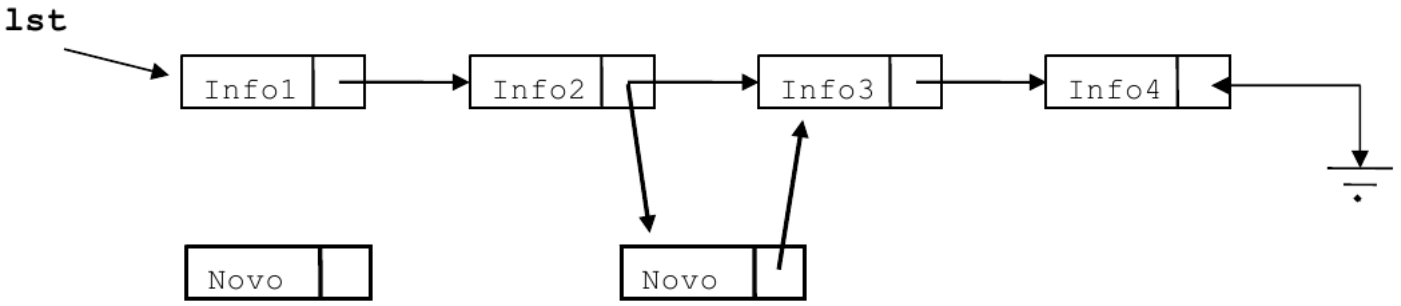
Função para liberar a lista

```
1    public void free(){
2        while (prim != null){
3            Nodo temp = prim.getProx();
4            prim = null;
5            prim = temp;
6        }
7    }
```

- Em Java, quando um objeto não é mais utilizado, a JVM é responsável por desalocar a memória que não é mais utilizada.
- No entanto, em outras linguagens de programação o programador deve explicitamente liberar a memória consumida pela variável desnecessária.

Manutenção da lista ordenada

Função de inserção percorre os elementos da lista até encontrar a posição correta para a inserção do novo.

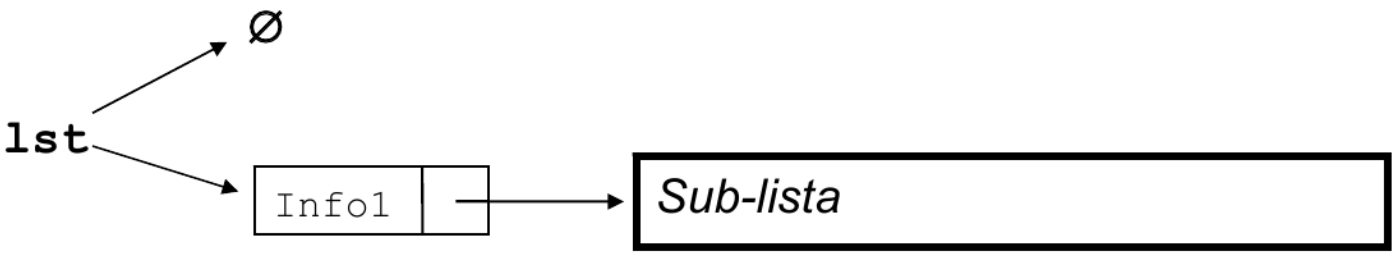


```
 1    public void addOrdenado(int i){
 2        Nodo novo;
 3        /*objeto para o elemento anterior*/
 4        Nodo anterior = null;
 5        /*objeto para percorrer a lista*/
 6        Nodo p = prim;
 7
 8        /*procura elemento na lista, guardando anterior*/
 9        while(p != null && p.getInfo() < i){
10            anterior = p;
11            p = p.getProx();
12        }
13
14        /*cria novo elemento*/
15        novo = new Nodo();
16        novo.setInfo(i);
17
18        /*encadeia o elemento*/
19        if(anterior == null){ /*inseri o elemento no inicio*/
20            novo.setProx(prim);
21            prim = novo;
22        }else{ /*inseri elemento no meio da lista*/
23            novo.setProx(anterior.getProx());
24            anterior.setProx(novo);
25        }
26    }
```

Definição recursiva de lista

Uma lista é:

- uma lista vazia, ou;
- um elemento seguido de uma (sub-)lista.



Exemplo: função recursiva para imprimir uma lista

- se a lista for vazia, não imprima nada
- caso contrário,
  - ⋆ imprima a informação associada ao primeiro nó, dada por $prim.getInfo()$
  - ⋆ imprima a sub-lista, dada por $prim.getProx()$, chamando recursivamente a função.

Função imprime recursiva

```
1  public void printRecursivo(Nodo n){
2      if(!isEmpty(n)){
3          /*imprime o primeiro elemento*/
4          System.out.println(n.getInfo());
5          /*imprime a sub-lista*/
6          printRecursivo(n.getProx());
7      }
8  }
```

Função imprime recursiva invertida

```
1  public void printRecursivoInvertido(Nodo n){
2      if(!isEmpty(n)){
3          /*imprime a sub-lista*/
4          printRecursivoInvertido(n.getProx());
5          /*imprime o elemento*/
6          System.out.println(n.getInfo());
7      }
8  }
```

Exemplo: função para retirar um elemento da lista

- retire o elemento, se ele for o primeiro da lista (ou da sub-lista)
- caso contrário, chame a função recursivamente para retirar o elemento da sub-lista

Função retira elemento recursiva

```
1   public Nodo removeRecursivo(Nodo n, int v){
2       if(!this.isEmpty(n)){
3           /*verifica se o elemento a
4            *ser retirado e o primeiro*/
5           if(n.getInfo()==v){ n = n.getProx();
6           }else{
7               /*retira da sub-lista*/
8               n.setProx(removeRecursivo(n.getProx(),v));
9           }
10      }
11      return n;
12  }
```

Igualdade de listas

boolean listasIguais(Nodo l1, Nodo l2)

Implementação não recursiva:

- percorre as duas listas usando dois ponteiros auxiliares:
  - ⋆ se duas informações forem diferentes então as listas são diferentes.
- ao terminar uma das listas ou as duas:
  - ⋆ se os dois ponteiros auxiliares são NULL então as duas listas tem o mesmo número de elementos e são iguais.

Listas iguais: não recursiva

```
1   public boolean listasIguais(Nodo l1, Nodo l2){
2       Nodo t1; /*objeto para percorrer l1*/
3       Nodo t2; /*objeto para percorrer l2*/
4       for(t1=l1, t2=l2;
5           t1 != null && t2 != null;
6           t1=t1.getProx(), t2=t2.getProx()){
7
8           if(t1.getInfo() != t2.getInfo())
9               return false;
10      }
11      return true;
12  }
```

Igualdade de listas

boolean listasIguais(Nodo l1, Nodo l2)

Implementação recursiva:

- se as duas listas dadas são vazias então são iguais
- se não forem ambas vazias, mas uma delas é vazia, então são diferentes
- se ambas não forem vazias, teste:
  - ⋆ se informações associadas aos primeiros nós são iguais e
  - ⋆ se as sub-listas são iguais.

Listas iguais: recursiva

```
1  public boolean listasIguaisRec(Nodo l1, Nodo l2){
2      if(l1 == null && l2 == null){
3          return true;
4      }else if(l1 == null || l2 == null){
5          return false;
6      }else
7          return
8            ((l1.getInfo()==l2.getInfo())
9             &&
10           listasIguaisRec(l1.getProx(),l2.getProx()));
11 }
```

Listas de tipos estruturados

Lista de tipo estruturado:

- a informação associada a cada nó de uma lista encadeada pode ser mais complexa, sem alterar o encadeamento dos elementos;
- as funções apresentadas para manipular listas de inteiros podem ser adaptadas para tratar listas de outros tipos.

- o campo da informação pode ser representado por um objeto para uma estrutura, em lugar da estrutura em si.
- independente da informação armazenada na lista, a estrutura do nó é sempre composta por:
  - ⋆ um objeto para a informação e
  - ⋆ um objeto para o próximo nó da lista.

Listas de tipos estruturados

```
1  public class Nodo{
2      private Aluno al;
3      private Nodo prox;
4  }
5
6  class Aluno{
7      private String nome;
8      private String matricula;
9      private float n1;
10     private float n2;
11     private float n3;
12 }
```

Listas heterogêneas

```
1   public class Nodo{
2       private FormasGeometricas fg;
3       private Nodo prox;
4   }
5
6   public class abstract FormasGeometricas{
7       private float b;
8       private float h;
9       public abstract float calculaArea();
10  }
11
12  public class Retangulo extends FormasGeometricas{
13      public float calculaArea(){
14          return b*h;
15      }
16  }
17
18  public class Triangulo extends FormasGeometricas{
19      public float calculaArea(){
20          return b*h/2;
21      }
22  }
```

Tópicos Complementares

Tópicos Complementares

- Listas Circulares
- Listas Duplamente Encadeadas

Material de **consulta**

- Capítulo 10 do livro: "Introdução a Estruturas de Dados" do Waldemar Celes, Renato Cerqueira e José Lucas Rangel.

Material de **referência**

- Capítulo 10 do livro: "Introdução a Estruturas de Dados" do Waldemar Celes, Renato Cerqueira e José Lucas Rangel.
- Imagens retiradas do site da disciplina de Programação II da PUC do Rio de Janeiro. http://www.inf.puc-rio.br/ inf1007/.