Curso de Engenharia da Computação

Material Disciplina Estrutura de dados – II

Estruturas de Lista Encadeada Circular e Lista de Salto.

Parte-II

Prof. Wagner Santos C. de Jesus

wsantoscj@gmail.com

Conceito de Lista Encadeada Circular

Conceito Lista Encadeada Circular

Tem o mesmo tipo de nodos (células) que uma lista encadeada simples. Isto é, cada nodo em uma lista encadeada circular tem um ponteiro para o próximo nodo e uma referencia para um elemento, entre tanto não existirá os extremos, superior e inferior.

Algoritmo Lista Circular

Declaração de elementos a serem inseridos:
Alocação da memória para o novo elemento.

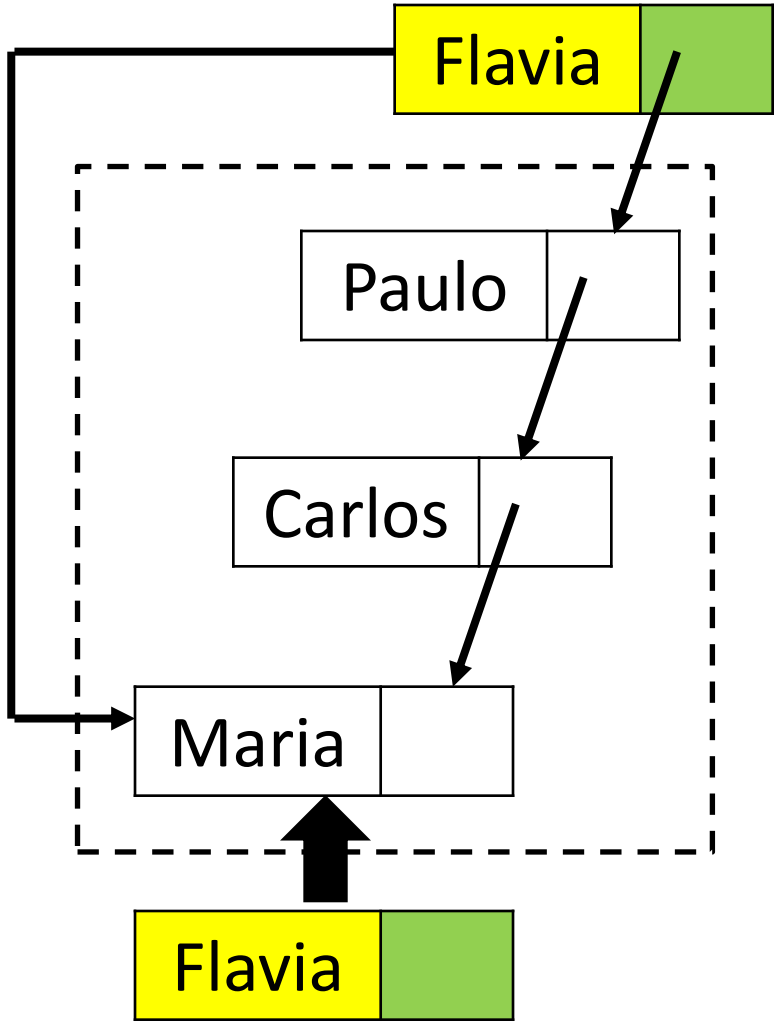
Preenchimento do conteúdo do campo de dados.

A atualização dos ponteiros com alvo no primeiro e no último elemento.

Caso especial: em uma lista com um único elemento, o primeiro também é o último.

Atualização do tamanho da fila.

Disposição de Lista Circular



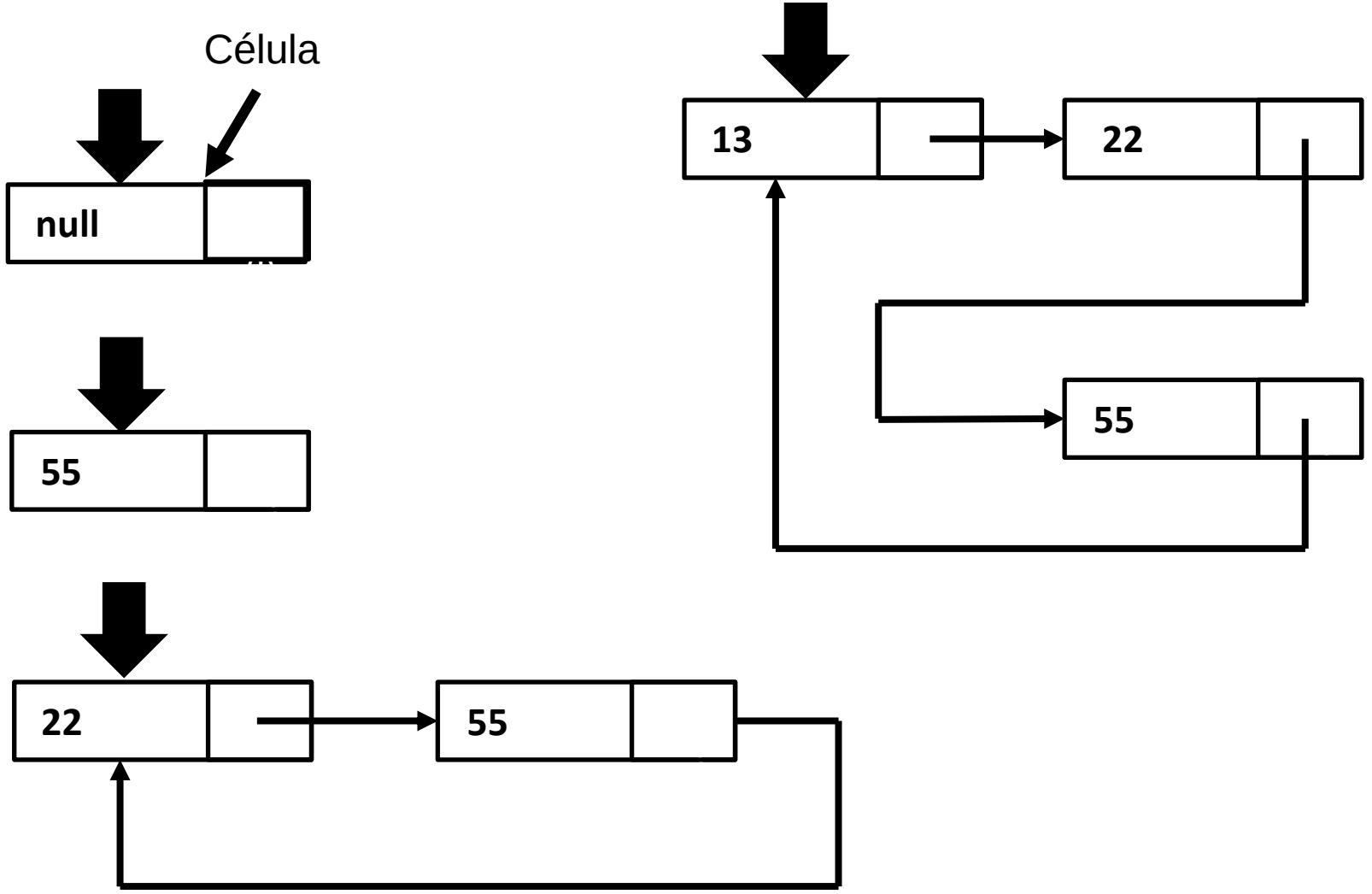
Ultimo elemento passa a ter prioridade de acesso ao se resgatar um elemento da lista.

Operações em Listas Encadeadas Circular

Operações principais de uma Lista Circular

Operações	Descrição
adiciona()	Insere um nó novo, v, após o cursor, se a lista está vazia, então v torna-se o cursor e o ponteiro aponta para si mesmo.
remove()	Remove o nodo v que se encontra imediatamente após o cursor (não o cursor propriamente dito, a menos que ele seja o único nodo); se a lista ficar vazia, o cursor é definido como **null.**
proximo()	Avança o cursor para o próximo nodo da lista.

Funcionamento

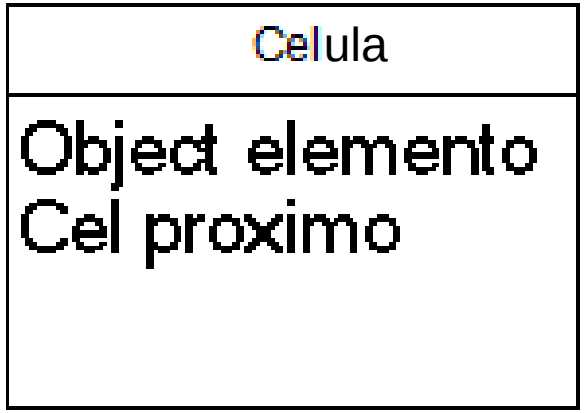


Implementação de Lista Encadeada Circular

Implementação da classe (nó) Célula

O primeiro passo é criar, uma estrutura de dados nó, onde o elemento da lista ficará armazenado, em seguida aponta para próxima posição nula.

Criação de uma Célula (Nodo)

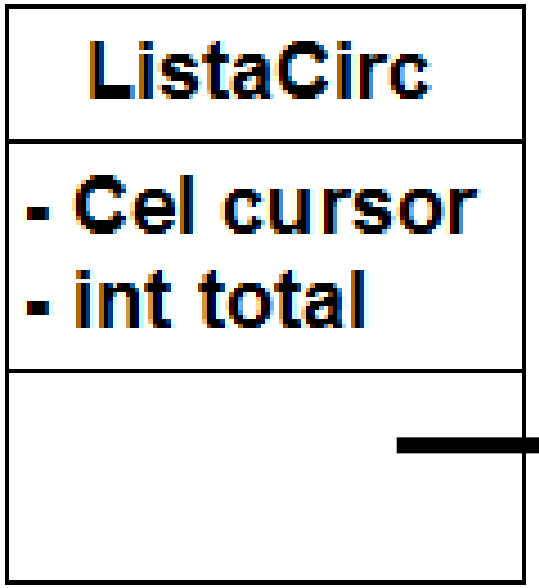



```
classe Celula {
      Objecto elemento;
      Celula proximo;
      Celula(Objecto elemento){
          this.elemento = elemento;
          this.proximo = null;
      }
}
```

Operações de manutenção da lista Circular.

- Navegar para próximo elemento da lista.
- Capturar um elemento corrente na lista.
- Verificar posição elemento cabeça da lista.
- Avança para um elemento da lista.
- Adicionar um novo elemento na lista.
- Remover um elemento da lista.
- Tamanho da lista.

Classe de operações da lista circular



Operações da lista circular:

proximo()

getelemet()

ultelement()

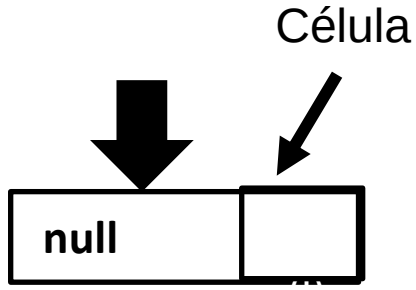
avanca()

adiciona()

remove()

tamanho()

Estrutura da classe (ListaCirc)



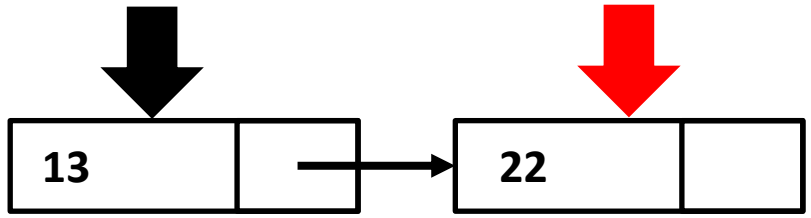

```
class Listacircular {
        privado Celula cursor;
        privado int total;

        publico Listacircular() { cursor=null; }
        ……
}
```

Implementação dos métodos da classe Lista Circular

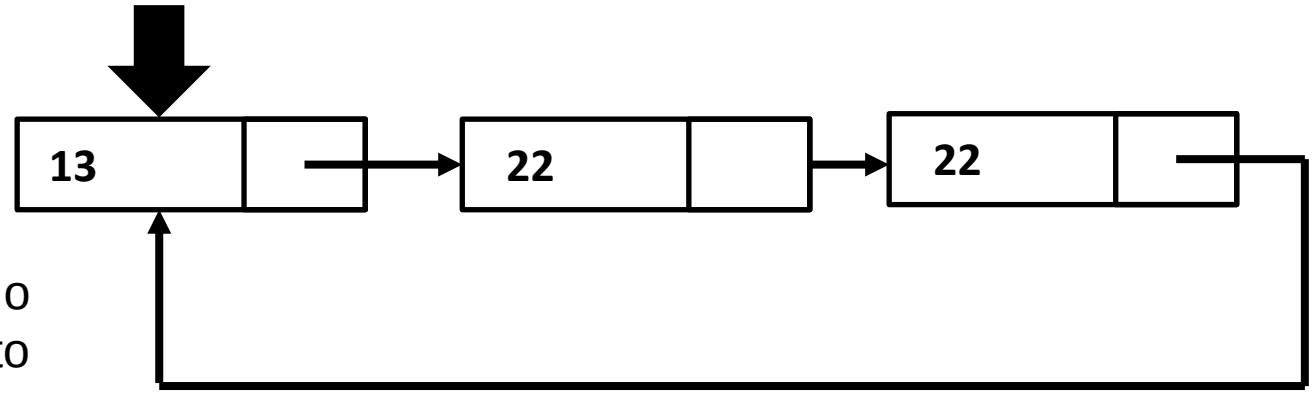
Desloca para próximo elemento

```
proximo() {
    cursor = cursor.proximo
}
```



Retorna um elemento da lista

```
Objecto getelement() {
        returno cursor.elemento
}
```



Captura sempre o ultimo elemento inserido na lista.
13

Tamanho de uma Lista Circular

```
inteiro tamanho() {
        retorno total;
    }
```

Retorna positivo caso seja o último elemento

```
booleano ultelement() {
        returno  (cursor == cursor.proximo);
    }
```

No caso de uma remoção de elementos o elemento cabeça não será removido, e o método ultelement irá retornar (verdadeiro).

Avança para o próximo elemento

```
vazio avanca(inteiro posicao) {
        para i de 1 ate posição faça
            this.proximo()
}
```

Adiciona elemento na Lista Circular

```
vazio adiciona(Objecto elemento){
        Celula nodo = new Celula(elemento);
        se (cursor==null) {
            nodo.proximo = nodo;
            cursor = nodo;
        } else {
             nodo.proximo = cursor.proximo;
             cursor.proximo = nodo;
             cursor = nodo;
         }
         total = total + 1
    }
```

Remove dados da Lista circular

```
vazia remove() {
        se(cursor!=null){
            se(cursor == cursor.proximo)
                cursor = null
             senão
                cursor.proximo = cursor.proximo.proximo
                total = total -1
        }
    }
```

Visualização Linear da Lista Circular

```
public String toString(){
        if(cursor == null) return "[]";
        String s = "["+this.getelement();
        Cel velhocursor = cursor;
        for(this.proximo();velhocursor!=cursor;this.proximo())
          s += ","+this.getelement();
          return s+ "]";
    }
```

Lista de Saltos (Skip List)

Histórico Lista Skip

Em 1990, William Pugh criou o que se denomina lista Skip, que permitem operações de pesquisa inserção e remoção mais eficientes do que a listas ligadas simples e biligadas.

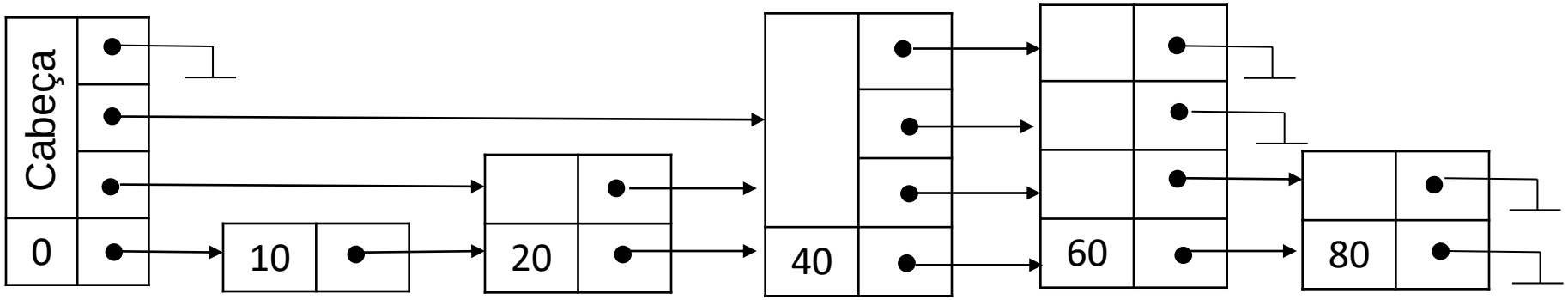
Conceito funcional de Lista Skip

Uma lista skip vem a ser uma lista ligada simples, em que cada nó tem um número variável de ligações, sendo que cada nível de ligação implementada uma lista ligada simples constituída por um número diferente de nós, ou seja uma estrutura de listas paralelas, todas terminadas com referencia nula.

Características Lista Skip

- Cabeça da lista nó extra que armazena um valor (v < todos elementos da lista).
- Número máximo de níveis da lista (5 e 30).
- Nível ativo atual que normalmente é zero.
- A lista Skip fica ordenada de forma crescente.

Estrutura da Lista Skip

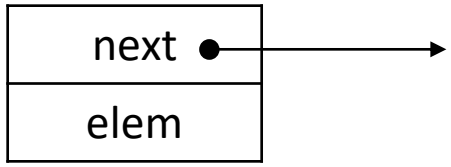


Ordenada por ordem Crescente

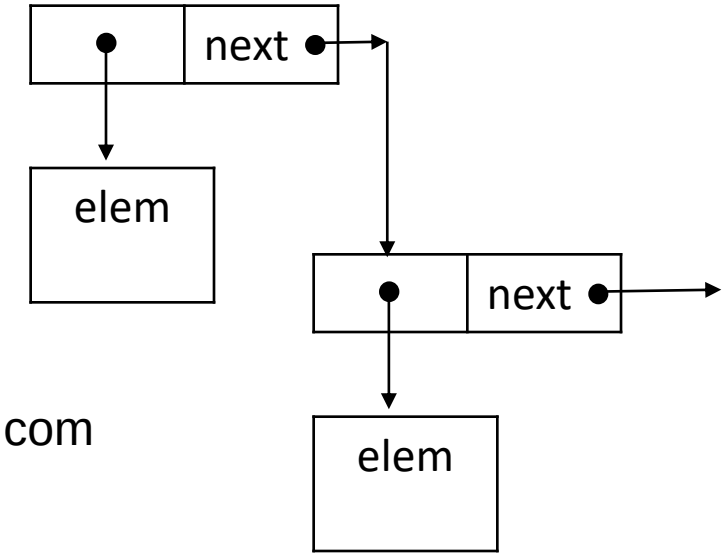
Conformação do nós da Skip list

No caso da esquerda da lista temos um nó compactado que armazena um elemento de um tipo de dados primitivo, enquanto que no caso da direita temos um nó decomposto que armazena um elemento de tipo de dados referencia. Como o número de referencias é variável de nó para nó, então o atributo **next** não é uma referencia para nó seguinte, mas sim uma sequencia de referencia para o nó seguinte.

Exemplo

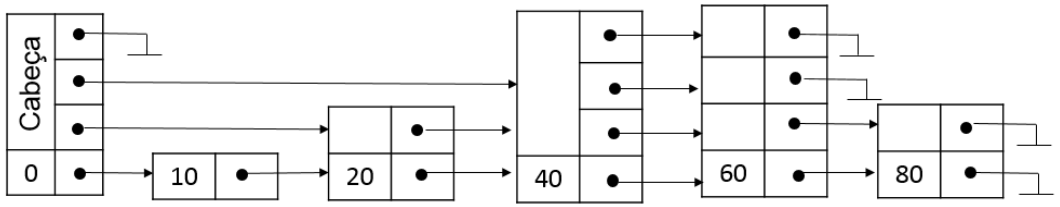


SkipNode[] next;
Tipo elem

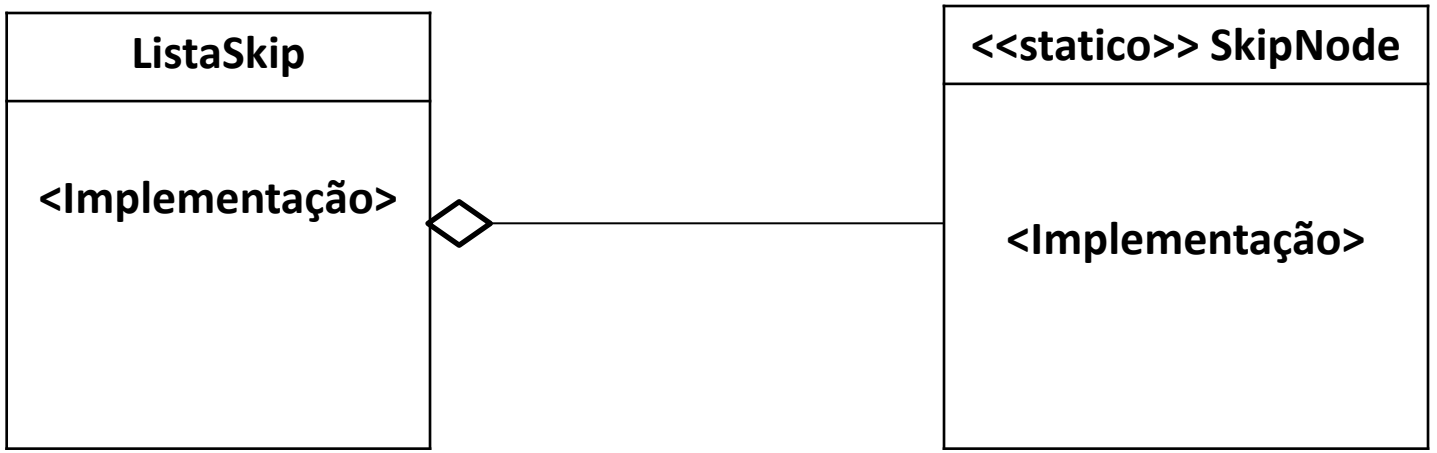


Estabelece a sequencia de elementos, com seus níveis intermediários.

Implementação de uma lista Skip



Classe ListaSkip



Conceito de agregação: Neste caso, o objeto da classe SkipNode, será parte do objeto da classe ListaSkip.

Estrutura Base Lista

```
classe ListaSkip {
        privado statico SkipNode {
                .....
            }
}
```

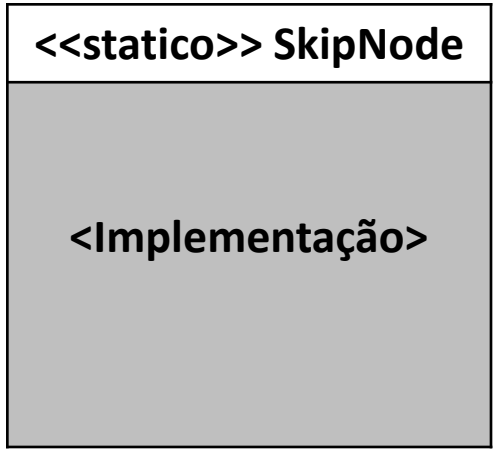
Construtor de nós da lista

```
publico SkipNode (inteiro pVal, inteiro pLev) {
        elem = pVal;
        next = new SkipNode[pLev];
}
```

pVal – Recebe o elemento dados que será armazenado na lista.

pLev – Números de nós que serão criados.

Implementação da Classe Nó(Célula)




```
privado statico classe SkipNode {
        publico SkipNode[] next;
        publico int elem;
        publico SkipNode (inteiro pVal, inteiro pLev) {
            elem = pVal;
            next = new SkipNode[pLev];
        }
  }
```

Propriedades da classe SkipList

privado SkipNode head;

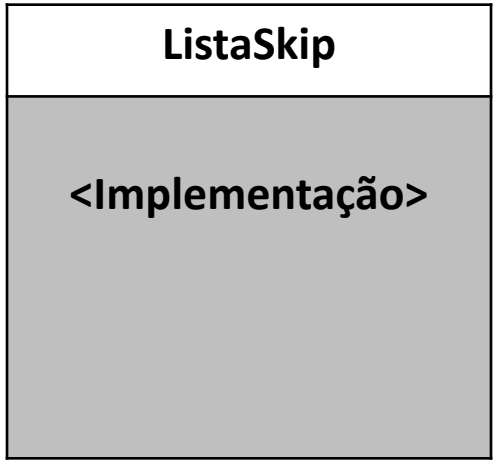
privado inteiro maxLevel;

privado inteiro actualLevel;

Propriedades	Descrição
head	Representa a cabeça da lista.
maxLevel	Representa o número máximo de níveis da lista.
actualLevel	Representa quantidade de níveis atual da lista.

Implementação do Construtor ListaSkip

```
publico ListaSkip(inteiro pVal, inteiro pMaxLevel){
        head = new SkipNode(pVal, pMaxLevel);
        maxLevel = pMaxLevel;
        actualLevel = 0;
        total = total + 1;
  }
```



Verificando Lista Vazia

Uma lista skip está vazia quando apenas existe o nó (célula) da cabeça, ou seja, quando todas as referencias deste nó são referencias nulas.

```
publico booleano isEmpty() {
    para i de 0; i < head.next.length; i++
        se (head.next[i] != null)
            return false;
        return true;
    }
```

(*)Processo de Recursão

Conceito Geral

Recursividade é um termo usado de maneira mais geral para descrever o processo de repetição de um objeto de um jeito similar ao que já fora exibido.

Conceito Computacional

Vem a ser processo pelo qual um procedimento realiza chamadas sequênciais dele mesmo.

Uma imagem recursiva

Foto: courtesia do Prof. Jim Bryan, Univ. of BC, Canada

Relações de recorrência

Matematicamente é comum definirmos sequências de números ou funções usando recursão. Exemplo Fatorial

$$f(n) = \begin{cases} 1 & \text{se } n = 1 \\ f(n-1) \times n & \text{se } n > 1 \end{cases}$$

Exemplo de função recursiva

```
int fatorial(int n)
  {
   if (n == 0)
      return 1;
      return (n * fatorial(n-1));
   }
```

Recursão e a pilha de execução (stack)

int x = fatorial(4)

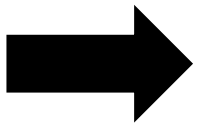
fatorial(4)	*fatorial(0)*
fatorial(3)	*fatorial(1)*
fatorial(2)	*fatorial(2)*
fatorial(1)	*fatorial(3)*
fatorial(0)	*fatorial(4)*

Elementos de uma função recursiva

- Condição de parada ou caso base ou caso trivial: é a parte da definição da função que não faz chamada recursiva.
- Chamada recursiva propriamente dita ou passo de recursão; deve resolver uma instância menor do mesmo problema.
- Processamento de apoio ou processamento complementar: demais processamentos que acompanham e/ou utilizam o que resulta da chamada recursiva.

Exemplo de função recursiva

```
int fatorial(int n)
  {
   if (n == 0)
      return 1;
      return (n * fatorial(n-1));
   }
```



Condição de parada

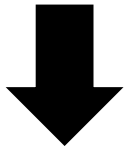
Exemplo de função recursiva

```
int fatorial(int n)
  {
   if (n == 0)
      return 1;
      return (n * fatorial(n-1)  );
   }
```

Chamada recursiva a uma instância menor

(*) Exemplo de função recursiva

```
int fatorial(int n)
  {
   if (n == 0)
      return 1;
      return(  n * fatorial(n-1)  );
   }
```

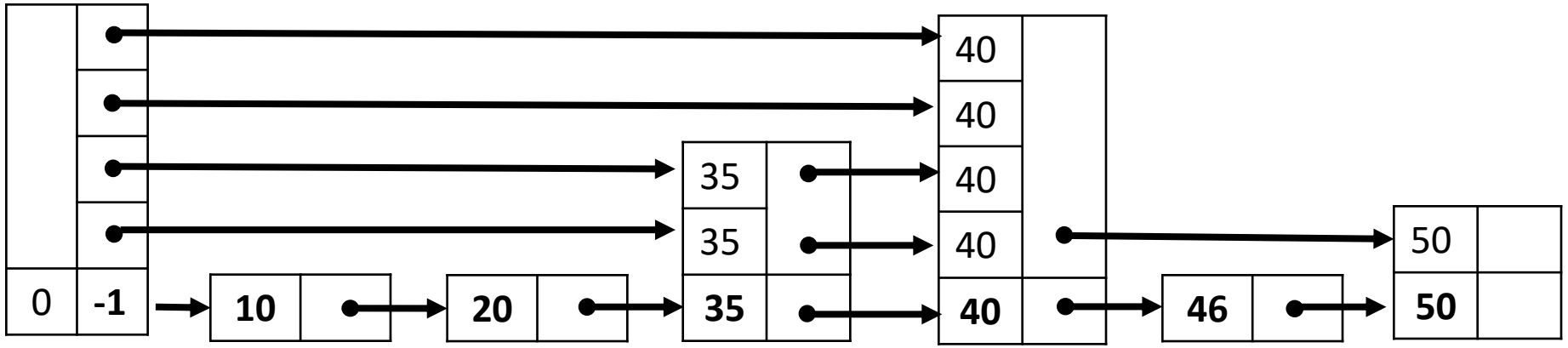


Processamento de apoio

Inserindo um elemento em uma lista Skip

O algoritmo de inserção do nó na lista analisa recursivamente todas as lista ligadas, desde o nível ativo atual até o nível zero, altura em que o processo de atualização das ligações acaba e o elemento é efetivamente inserido na lista.

Contexto aleatório a cada inserção



Cada elemento será inserido mediante ao estabelecimento de um nível, que será inserido na sequencia, os níveis são descobertos de forma aleatória.

Método de Geração de Nível de nó

Esse algoritmo apresenta a proposta por Robert Sedgewick que determina aleatoriamente o nível i (com i ∈ [i..maxLevel]), com probabilidade $1/2^i$ Sempre que o nível gerado é superior ao nível atual da lista a variável actualLevel, será atualizada.

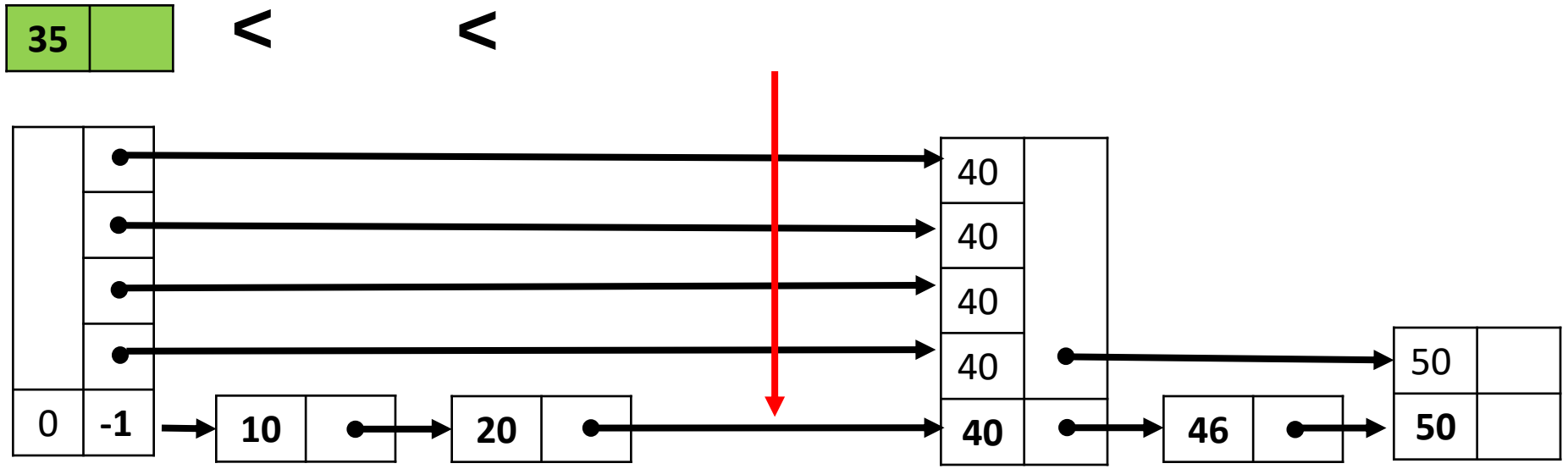
Determina o nível da lista (Aleatoriamente)

```
privado inteiro rand() {
        inteiro i,j;
        flutuante t = Math.random();
        para(i=1,j=2; i< maxLevel; i++, j+=j)
            if(t*j>1.0) break;
        se(i > actualLevel) actualLevel = i;
        return i;
    }
```

Inserção de um elemento na lista de Salto

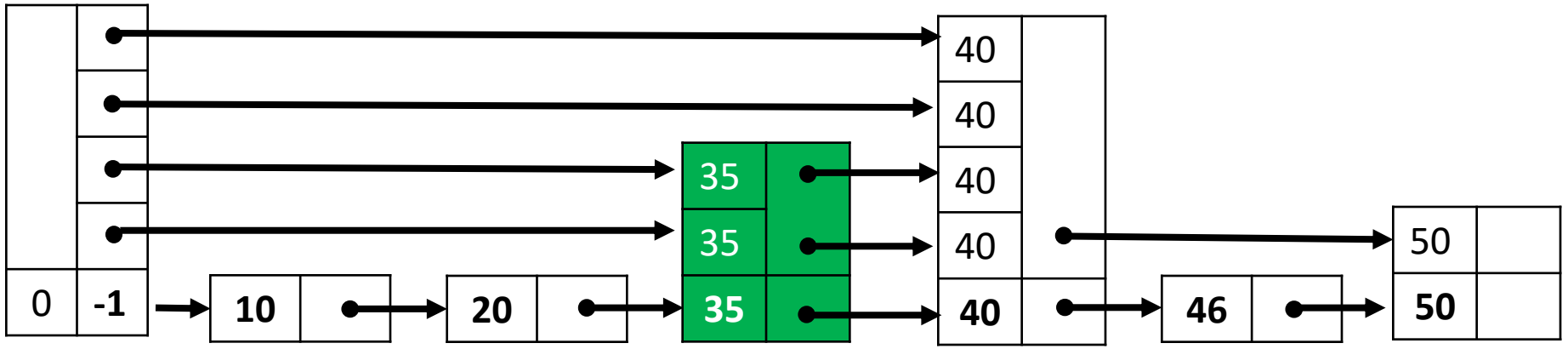
```
publico vazio insert(inteiro pElem){
        int rnd = rand();
        SkipNode ins = new SkipNode(pElem,rnd);
        recInsert(head,ins,actualLevel-1);
        total = total + 1;
    }
```

Inserção de elementos na Lista



Realiza o teste, em cada nível até que encontre um elemento que seja menor que o elemento do nó pesquisado, quando essa tarefa for realizada, então os elemento será enviado para o nível zero sendo inserido.

Resultado da Inserção

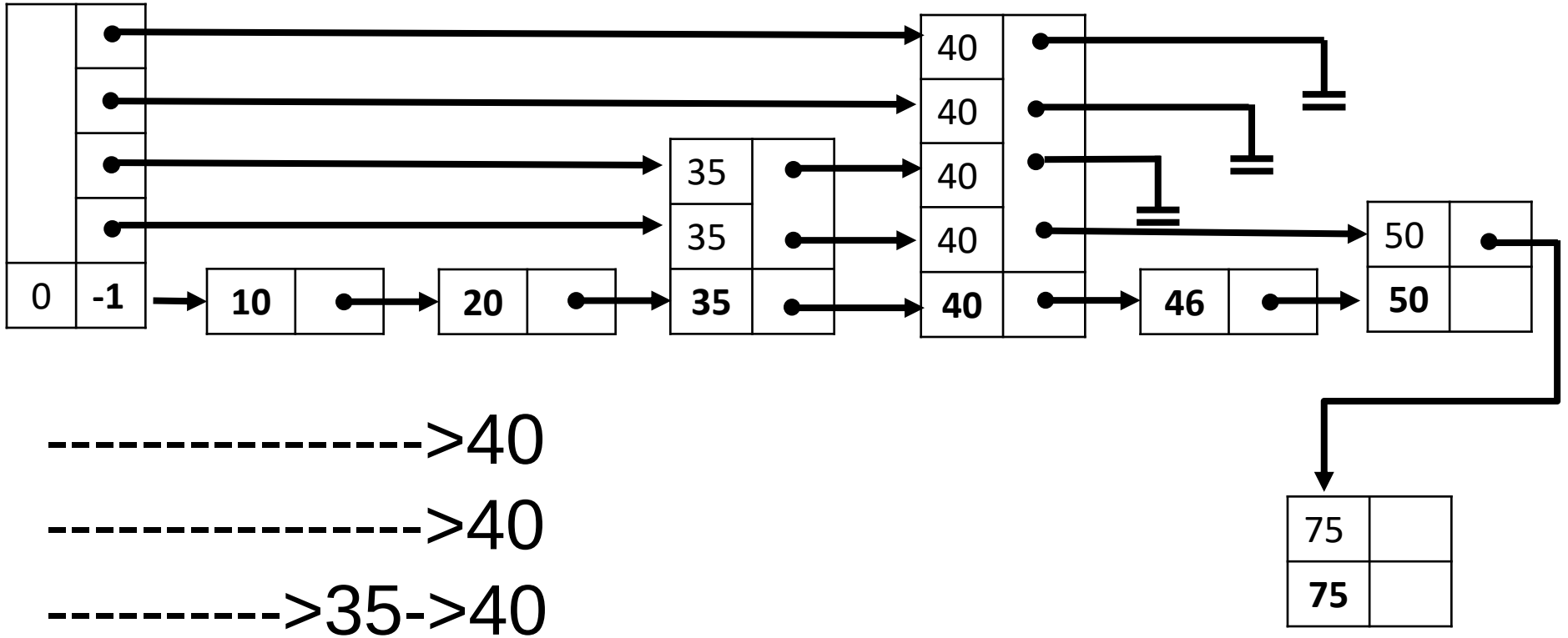


Método de Inserção de elementos

```
privado statico vazio  recInsert(SkipNode phead, SkipNode pNew, inteiro pLev) {
        se(phead.next[pLev] == null || pNew.elem < phead.next[pLev].elem)
        {
            se(pLev < pNew.next.length){
                pNew.next[pLev] = phead.next[pLev];
                phead.next[pLev] = pNew;
            }
            se(pLev == 0) return;
              recInsert(phead, pNew, pLev-1);
        } senão
            recInsert(phead.next[pLev], pNew, pLev);
    }
```

Exibindo Dados de uma Lista Skip Implementação do Método (toString())

Criando a sequência de Saída



---------->40

---------->40

---------->35->40

---------->35->40--->50->75

-1 10 20 35 40 46 50 75

```
public String toString() {
    se(isEmpty())
       return "Lista vazia";
    SkipNode node, test;
    String str = "Lista Skip\n";
    para(int i=maxLevel-1;i > 0; i--)
    {
        node = head;
        se(node.next[i] == null)
          str += "\n";
        senão {
         str += "> "+node.next[i].elem;
         test = head.next[0];
         para(node = node.next[i]; node != null; node = node.next[i]) {
            para(;test!= node; test= test.next[0])
               str += "\t";
               se (node.next[i]== null)
                  str +="\t*";
               senão
                 str += " \t> "+node.next[i].elem;
                 test = test.next[0];
            }
```

Mostra a hierarquia de níveis já sorteadas, com seus respectivos apontamentos.

```
    str += "\n";
        }
    }

str += head.elem;
para(node=head.next[0];node!=null;node=node.next[0])
        str += "\t"+node.elem;

    return str+"\t*\n";
}
```

Mostra os elementos do nível zero da lista “skip”.

Métodos Busca em uma Lista Skip

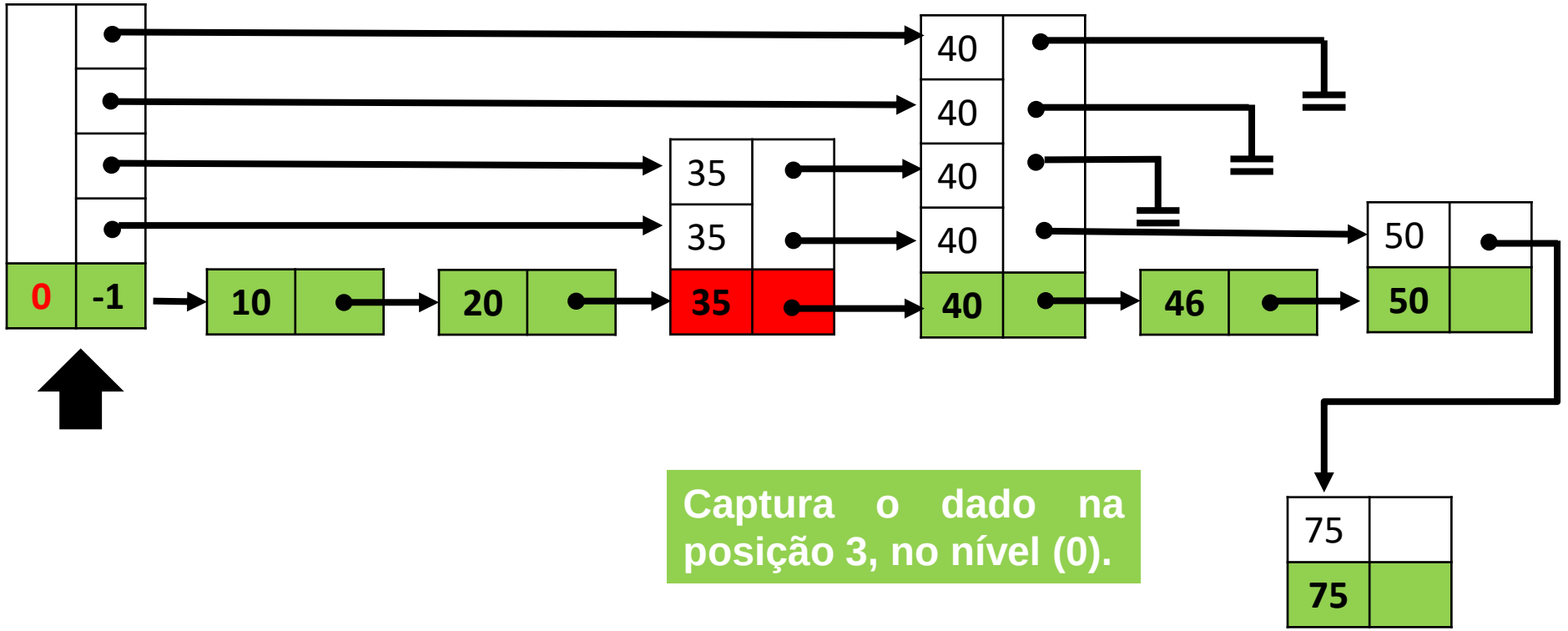
Conceito de deslocamento de busca de elementos.

Para capturar um elemento em uma lista skip, deve-se deslocar a referencia do objeto, utilizando apenas os deslocamentos em nível (0).

Procedimentos

- Verificar se o índice da posição especificada é > zero.
- Deslocar o ponteiro até a posição escolhida.
- Retornar o valor da posição do ponteiro.

Pesquisa somente em nível zero



Captura o dado na posição 3, no nível (0).

Captura um elemento da lista

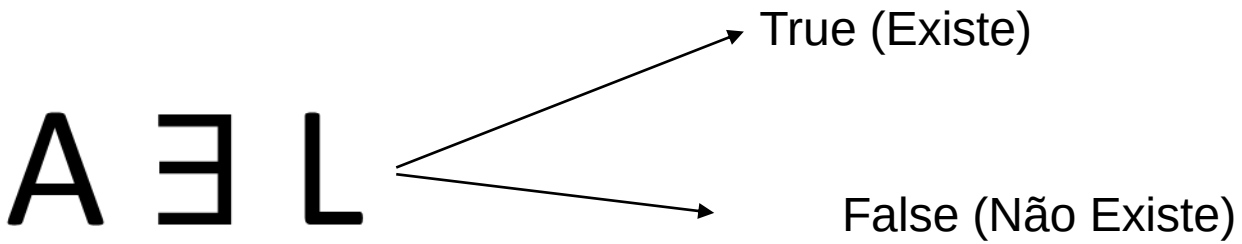
```
publico inteiro getelement(inteiro posicao) {
        SkipNode node;
        node = head;
        se(posicao == 0)
          return head.elem;
        para(int i=0;i<=posicao-1;i++)
            node=node.next[0];

    return node.elem;
}
```

Realizando verificação da Existência de um dado na Lista Skip

Relação Matemática

Sabe-se que **A** é o elemento a ser procurado no conjunto L, onde (L) lista de onde existe uma estrutura de dados.



Implementação

```
publico booleano  search (inteiro pElem) {
        returno recSearch(head, pElem, actualLevel-1) != null;
    }
```

Retorna o resultado do método recSearch, verdadeiro caso sua resposta seja diferente de nulo.

Realiza busca Recursiva de elementos em cada nível.

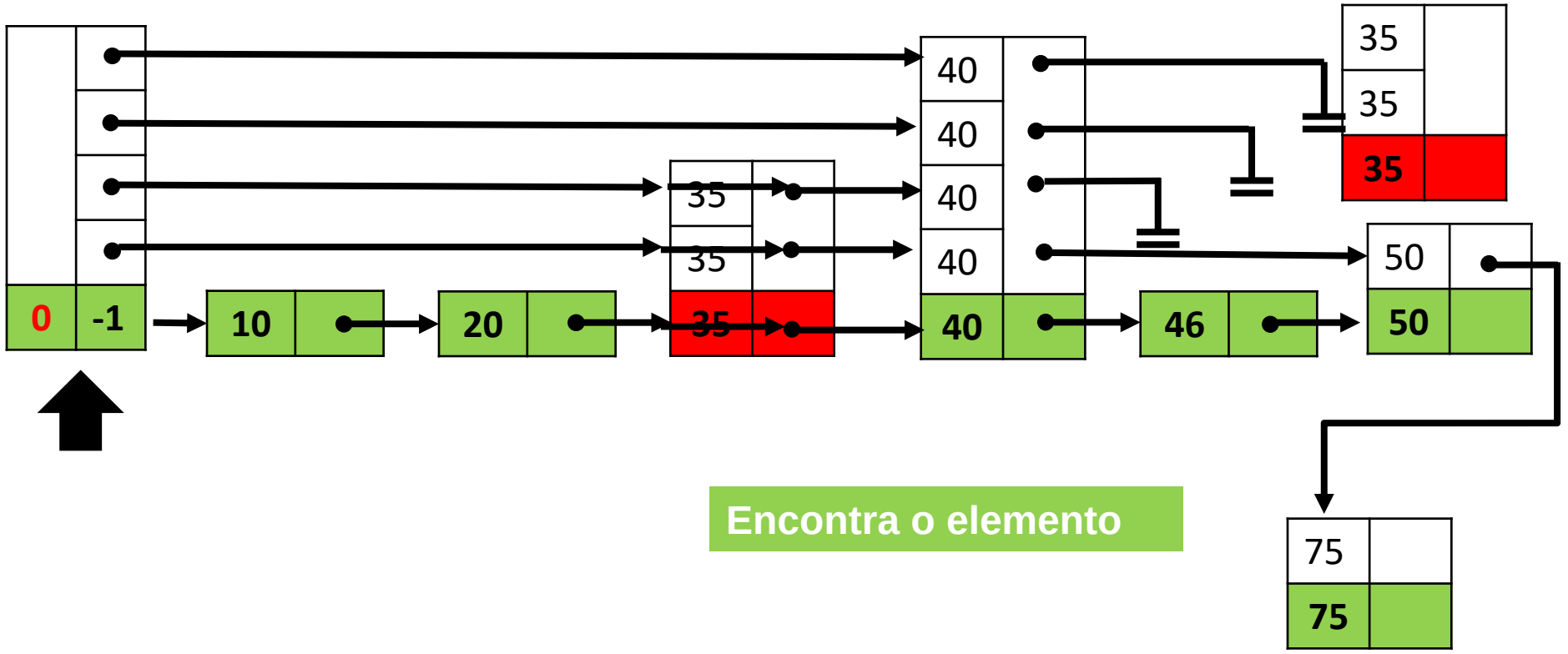
```
private static SkipNode recSearch(SkipNode pNode, int pElem, int pLev) {
        se(pElem == pNode.elem)
            return pNode;
         enquanto (pLev > 0 && pNode.next[pLev] == null)  faça
            pLev--;
        fim-enquanto
        se(pElem < pNode.next[pLev].elem)
        {
                se(pLev == 0) return null;
                return recSearch(pNode, pElem, pLev-1);
           }
           return recSearch(pNode.next[pLev],pElem,pLev);
    }
```

Eliminação de elementos em uma Lista Skip

Realiza a chamada do método recursivo de deleção.

```
publico vazio delete(inteiro pElem){
        se (this.search(pElem)) {
            recDelete(head,pElem,actualLevel-1)
            total = total -1
        }
    }
```

Pesquisa somente em nível zero



Apagando elemento da lista

```
privado static vazio recDelete(SkipNode pHead, int pElem, int pLev){
        enquanto (pLev > 0 && pHead.next[pLev] == null)  faça
            pLev--;
        se(pLev < 0 || pHead.next[0] == null)
           return;
        SkipNode node = pHead.next[pLev];
        se(node.elem >= pElem){
            se(node.elem == pElem){
                pHead.next[pLev] = node.next[pLev];
                se(pLev==0) return;
            }
            recDelete(pHead,pElem,pLev-1);
        }
        senão
          recDelete(pHead.next[pLev],pElem,pLev);
    }
```