

Relatório - Lista Dinâmica Duplamente Encadeada

1. Descrição do Algoritmo

1.1. Inserção

- Instancia um novo node com o valor numérico informado
- Através de duas variáveis temporárias, uma para o node antigo e uma para o atual, faz uma varredura na lista em busca do último node inserido com valor menor ou igual ao valor informado pelo usuário.
- Efetuada a inserção, aumenta o tamanho da lista.

1.2. Remoção

- Verifica se a lista está vazia ou se o elemento informado não foi encontrado
- Através de duas variáveis temporárias, uma para o node antigo e uma para o atual, faz uma varredura na lista em busca do último node inserido com valor menor ou igual ao valor informado pelo usuário.
- Se encontrado:
 - Verifica se o node antigo existe
 - se sim, seta o node a sua direita (next) como o próximo do node atual e verifica se o node a direita do node atual existe, se sim, seta o node atual como antigo, senão, seta o último elemento da LDDE como o node antigo
 - caso contrário (remoção do primeiro elemento), seta o primeiro node da LDDE como o próximo do node atual e verifica se o node a direita do node atual existe, se sim, seta o atributo do node anterior ao primeiro elemento como null, caso contrário, seta o último elemento da LDDE como null (remoção em LDDE de tamanho unitário)
 - Decrementa o tamanho da lista
- Retorna verdadeiro

1.3. Busca

- Verifica se a lista está vazia, se sim, retorna falso
- Inicializa uma variável para node atual (começa como first), e através de um laço faz uma varredura na lista em busca do último node inserido com valor menor ou igual ao valor informado pelo usuário.

1.4. Limpa

- Setta os atributos referentes ao primeiro e último node como null e zera o tamanho da lista

2. Pseudo-código

```
class Node(value)
    initialize value to value parameter
    initialize previous to null
    initialize next to null
```

```
class DoublyLinkedList
    initialize first to null
```

initialize last to null

initialize size to zero

method insert (parameters: number to value): boolean

create new Node instance with the passed value

initialize previousNode to null

initialize currentNode to null

while currentNode is not null and currNode value is less than value parameter

 previousNode <- currNode

 currentNode <- currNode.next

endwhile

if previousNode is not null

 previousNode.next <- currentNode

otherwise

 first <- newNode

endif

if currentNode is not null

 previousNode <- newNode

otherwise

 last <- newNode

endif

newNode.next <- currentNode

newNode.previous <- prevNode

add one to size

return true

method remove (parameters: number to value): boolean

declare previousNode

declare currentNode

if the DLL is not empty and the value was found in the list

 return false

endif

while exists a node with the passed value in the list

 previousNode <- null

```
currentNode <- currentNode.next
```

```
while currentNode is not null and currNode value is less than value parameter
```

```
    previousNode <- currNode
```

```
    currentNode <- currNode.next
```

```
endwhile
```

```
if value of current node is equals to value parameter
```

```
    then if previousNode is not null
```

```
        currentNode <- currentNode.next
```

```
        if currentNode.next is not null
```

```
            currentNode <- previousNode
```

```
        otherwise
```

```
            last <- previousNode
```

```
        endif
```

```
    otherwise
```

```
        first <- currentNode.next
```

```
        if currentNode.next is not null
```

```
            first.previous <- null
```

```
        otherwise
```

```
            last <- null
```

```
    endif
```

```
    decrement one to size
```

```
endwhile
```

```
return true
```

method search (parameters: number to value): boolean

```
    if the list is empty
```

```
        return false
```

```
    endif
```

```
    initialize currentNode to first node
```

```
while currentNode is not null and currentNode.value is less than value parameter
    currentNode <- currentNode.next
endwhile

if currentNode is not null
    return if currentNode.value equals value parameter

return false
```

method clear(): void

```
set first to null
set last to null
set size to 0
```

3. Vantagens e desvantagens

3.1. Vantagens

- É uma estrutura bidirecional, diferentemente de uma Linked List comum, que só pode ser percorrida em uma direção
- Como o seu próprio nome diz, é uma estrutura dinâmica, ou seja, possui tamanho flexível, diferentemente de um array estático.
- Numa Linked List comum, para a operação de remoção, por exemplo, é necessário criar uma variável temporária para o nó anterior, na LDDE temos um atributo próprio para isso.
- Suas operações de inserção e remoção são fáceis, pois apenas exigem que alteremos as referências relacionadas ao nó a direita/esquerda

3.2. Desvantagens

- Todas as operações exigem a manutenção de um atributo extra, referente ao nó anterior, o que faz com que a LDDE exija memória extra e mais tempo para a realização das operações de inserção e remoção
- Os elementos não são armazenados de forma sequencial, diferente de um array, que permite acesso direto aos seus elementos necessitando apenas dos índices destes.