E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Estruturas de Dados e Algoritmos Fundamentais

CÓDIGO: 21046

DOCENTE: Paulo Shirley

A preencher pelo estudante

NOME: Mário Pedro Capela Rodrigues Carvalho

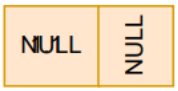
N.º DE ESTUDANTE: 2000563

CURSO: Licenciatura em Engenharia Informática

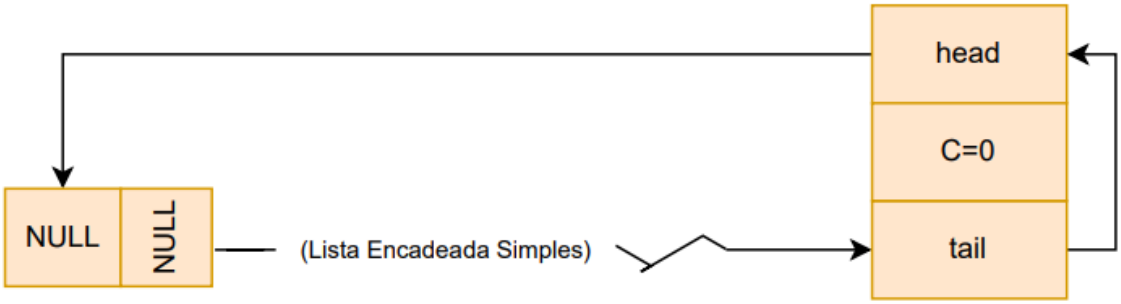
DATA DE ENTREGA: 17 de Abril de 2023

TRABALHO / RESOLUÇÃO:

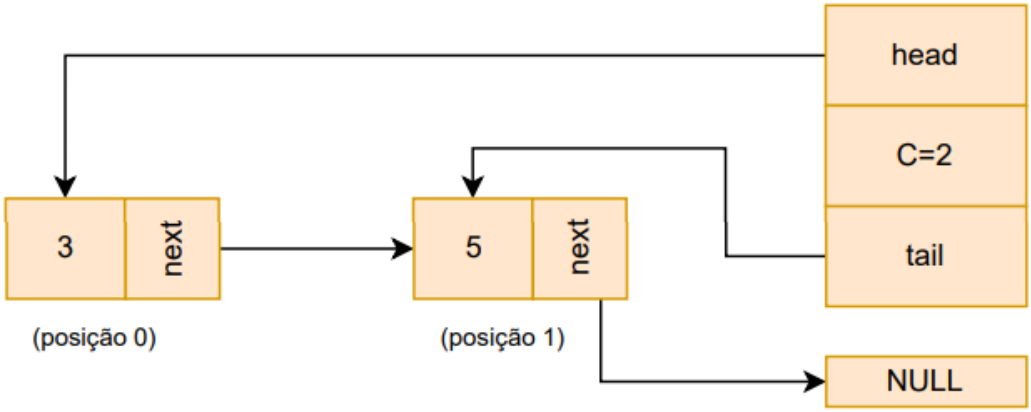
1. Tendo em conta que se trata de uma lista simplesmente ligada (*single linked list*), que armazena itens que são números, tendo em conta que um nó é representado da seguinte forma:



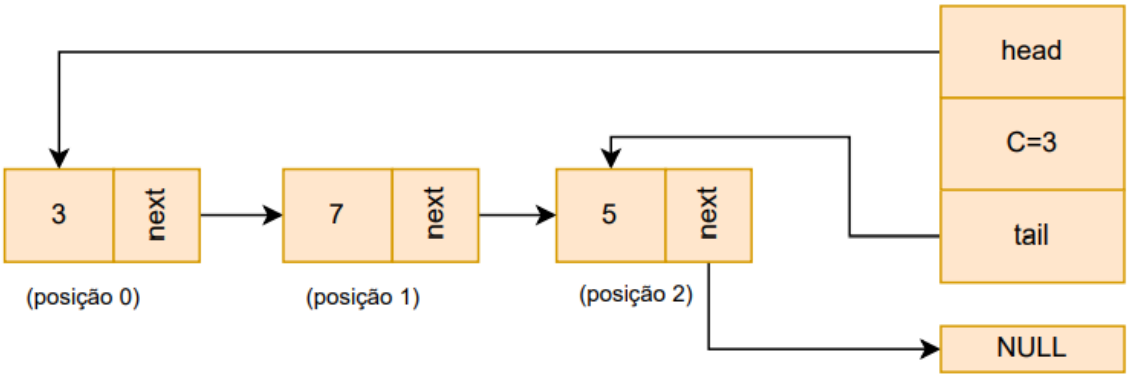
1.1- Perante um exemplo de uma lista dispõe de um apontador (*head*) para o primeiro nó da lista e um contador (n) com o número atual de nós da lista, e que esta tem um nó final (*tail*), numa fase inicial a sua representação é a seguinte:



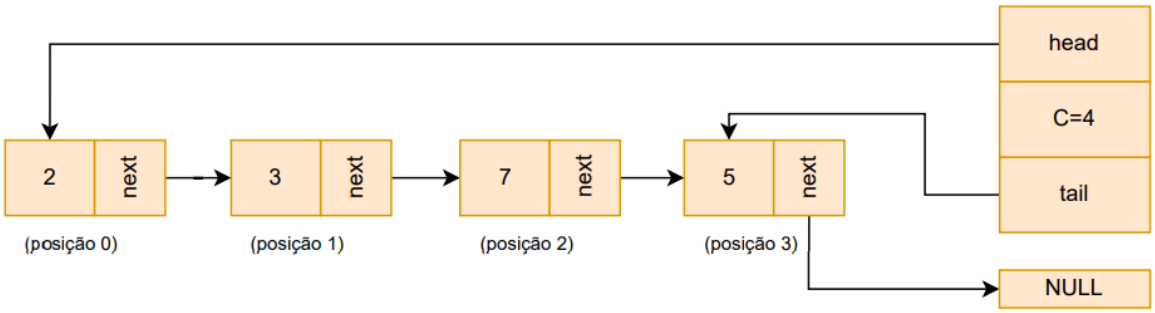
1.1.1- Ao inserir no fim da lista os itens 3, 5 por esta ordem, o diagrama final fica:



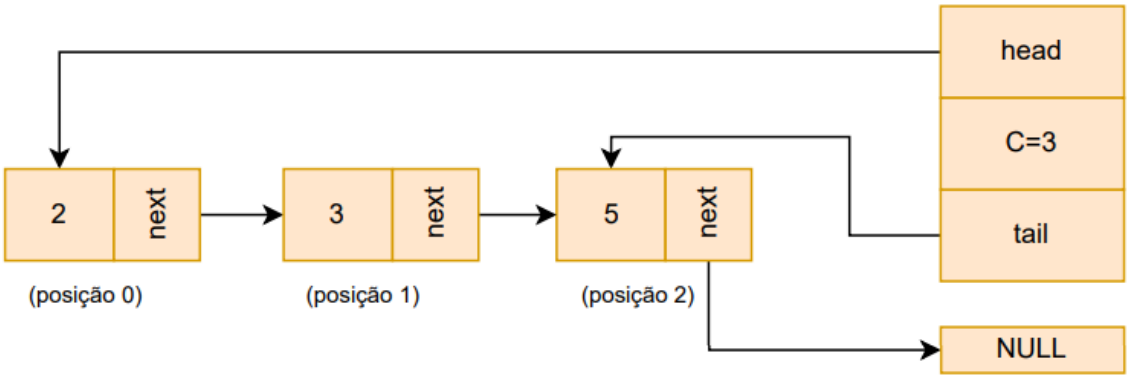
1.1.2- Inserindo na posição 1 o item 7, o diagrama final fica:



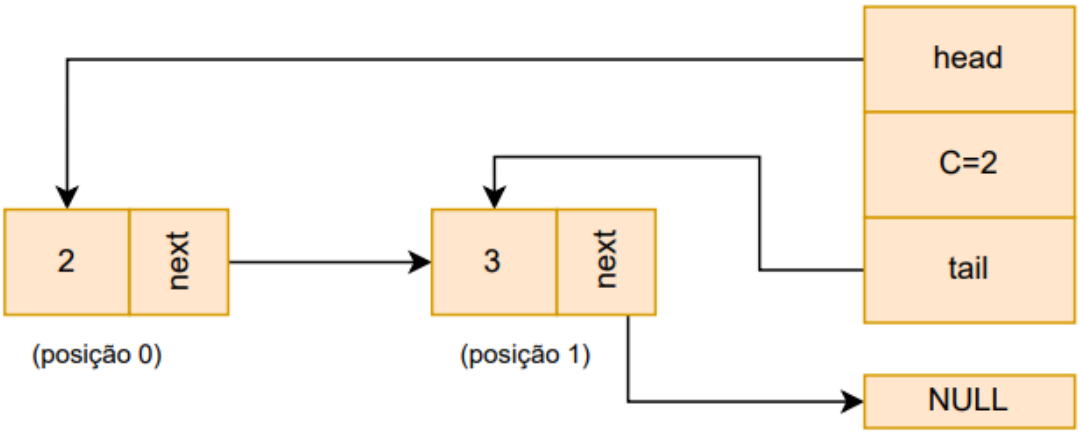
1.1.3- Inserir no início da lista o item 2, o diagrama final fica:



1.1.4- Ao remover da lista o item 7, o diagrama final fica:



1.1.5- E por fim, ao remover o último item da lista, tem-se como diagrama final:



1.2- Tal como pedido, no fim do e-fólio, encontra-se todo o código utilizado para implementar as especificações pedidas, com as seguintes funções:

```
void insert_0(string);
void insert_end(string);
void print_0(string);
void print_end(string);
void print(string);
void delete_0(string);
void delete_end(string);
void dim(string);
void clear(string);
```

```
void find(string, string);
void find_max(string, string);
void delete_pos(string, string);
void invert_range(string, string);
bool verificarListaVazia(string);
bool posicaoValida(string, int, int);
```

A percentagem de testes resolvidos com sucesso foi de: **100%** no código no meu computador. O que faz que toda a resolução de cada um destes exercícios está correta, pois acaba por imprimir e fazer tudo o que é suposto para cada função / método pedido.

1.3- A operação implementada que se recorre para executar a operação de remoção de um nó numa determinada posição da lista encadeada simples, ocorre em diferentes etapas, portanto, cada uma com complexidade do comando na notação "*Big-O*", diferente. Para melhor facilidade na compreensão do que consiste cada parte, identifica-se o que acontece e a complexidade respetiva da seguinte forma:

1. Numa fase inicial, procede-se à introdução de valores ("comando + argumentos") que são tratados pelo "*getline*", onde de seguida, o comando é verificado se é válido (seguindo o método da função "validarComando()") constando se ao comando introduzido existe um valor associado. Só nas linhas seguintes do código é que a lista é preenchida, com todos os argumentos que dela fazem parte. Complexidade O(1), pois, a lista encontrava-se vazia, e nada é devolvido, e com tempo de execução constante.
2. A fase seguinte, consiste em executar o comando, quando o valore em comando é validado. Neste momento, a "*string*" introduzida associada à variável "comando" passa por uma fase de controlo, dentro do mesmo método referido "executarComando()". Aqui, procede-se a uma comparação de comando com um conjunto de caracteres, que se for idêntico ao comando atribuído, o programa entra na função, que neste caso é "delete_pos". Esta função, encontra-se identificada no documento "isll.h", e implementada em "isll.cpp". Quando entra nesta função, a lista que já estava preenchida, continua com acessibilidade direta, e como mas mais nada é realizado nela, o tempo de execução continua constante e portanto, a complexidade contínua é O(1).
3. O passo seguinte, dentro do método "delete_pos", não atinge de maneira nenhuma a complexidade, pois, apenas se verifica em primeiro lugar (com auxilio ao método "verificarListaVazia()"), a existência de valores nesta lista. Se for falso, é

devolvido “false”, e então significa que existem valores na lista, e aqui, a complexidade continua de O(1). Neste momento o acesso é imediato, com valor devolvido, portanto, com um tempo de execução constante. O mesmo acontece com a verificação da posição para validar o argumento introduzido.

4. A manipulação da lista que ocorre nesta fase, já irá afetar a complexidade, que é O(n), pois, consiste na remoção do nó de uma lista. Nesta parte da função, após se criar um apontador, o programa percorre a lista até (“*tail*” e até) detetar o nó, cujo apontador atual encontra em "*next*”. A posição que se procura (dada por “pos”), que é encontrada no momento que o nó atual (de nome “inicio”) se encontra, aponta para o nó da lista “*next*” onde se encontra o valor correspondente a apagar (e portanto, o da posição respetiva a “*pos*”) que se encontra no nó associado. Em seguida, atualiza-se o apontador "*next*" do nó “anterior” para apontar para o primeiro nó seguinte que se encontra na lista (“atual->*next*”). Imediatamente a seguir ao nó “início” é apagado. Por esta razão, se recorreu à criação de dois nós temporários (“inicío” e “anterior”), para se poder alterar os locais de memória sem apagar os items. É importante notar que o tempo de execução deste algoritmo varia de acordo com o tamanho da lista (n), uma vez que é necessário percorrer todos os seus elementos. Assim, até terminar a leitura, a lista pode ter sido percorrida inteiramente, e por isso, grande parte do número de posições potenciais a serem removidas também tiveram de ser lidas. Por isso, a justificação para a sua complexidade diferente das fases anteriores.

Anexo - Testes:

Dadas as complexidades constatadas durante as tentativas de realizar o depósito dos ficheiros na plataforma para realizar os testes pedidos, ainda que tivessem sido tentadas todas as alternativas sugeridas no fórum, apresentam-se de seguida os testes realizados localmente, seguidos dos respetivos resultados

Teste 00:

```
insert_0      2
insert_0    -1
insert_end      5
print
Lista= -1 2 5
```

Teste 01

```
insert_0   8 1 9  7 5
insert_0    0
insert_end   1 2 3 4  5  6
print
Lista= 0 5 7 9 1 8 1 2 3 4 5 6
```

Teste 02

```
insert_end   -2 -1 0 1 2 3 4 5  6
print
Lista= -2 -1 0 1 2 3 4 5 6

print_0
Lista(0)= -2

print_end
Lista(end)= 6
```

Teste 03

```
clear
Comando clear: Lista vazia!

insert_0     3 6 4 8 6  1 7 12
print
Lista= 12 7 1 6 8 4 6 3

delete_0
print_0
Lista(0)= 7

delete_end
print_end
Lista(end)= 6

print
Lista= 7 1 6 8 4 6
```

Teste 04

```
insert_end   0 2 4 6 8 10 12 14 16 18 20
print
Lista= 0 2 4 6 8 10 12 14 16 18 20

find         8
Item 8 na posicao 4
find         0
Item 0 na posicao 0
find         20
Item 20 na posicao 10
delete_end
insert_end   18
insert_end   18
print
Lista= 0 2 4 6 8 10 12 14 16 18 18 18

find_max
Max Item 18 na posicao 9!
```

Teste 05

```
insert_end   0 2 4 6 8 10 12 14 16 18 20
print
Lista= 0 2 4 6 8 10 12 14 16 18 20

find         8
Item 8 na posicao 4
find         0
Item 0 na posicao 0
find         20
Item 20 na posicao 10
delete_end
insert_end   18
insert_end   18
print
Lista= 0 2 4 6 8 10 12 14 16 18 18 18

find_max
Max Item 18 na posicao 9!
```

Teste 06

```
insert_end  0 1 2 3  4  5  6 7 8 9
print
Lista= 0 1 2 3 4 5 6 7 8 9

delete_pos  4
print
Lista= 0 1 2 3 5 6 7 8 9

delete_pos  0
print_0
Lista(0)= 1

delete_pos  7
print_end
Lista(end)= 8

print
Lista= 1 2 3 5 6 7 8

delete_pos  70
Comando delete_pos : Posicao invalida!
```

Teste 07

```
insert_end   -1 1 3 5 7 9 10 11 12 13 15 17 -17
print
Lista= -1 1 3 5 7 9 10 11 12 13 15 17 -17

invert_range    5  9
print
Lista= -1 1 3 5 12 11 10 9 7 13 15 17 -17

invert_range    10  12
print
Lista= -1 1 3 5 12 11 10 9 7 17 15 13 -17
```

Teste 08

```
clear
Comando clear: Lista vazia!

insert_0      3 6 4 6 8 4 2 9 6 4 6 3
insert_end   -1 -5 12 3 45 -23 45  30  30
print
Lista= 3 6 4 6 9 2 4 8 6 4 6 3 -1 -5 12 3 45 -23 45 30 30

dim
Lista tem 21 itens

delete_0
delete_0
delete_end
find_max
Max Item 45 na posicao 14!

delete_pos 10
print
Lista= 4 6 9 2 4 8 6 4 6 3 -5 12 3 45 -23 45 30
```

Teste 09

```
clear
Comando clear: Lista vazia!

insert_end   -1 -5 12 3 45 -23 45 30 30
insert_end   -1 -5 12 3 45 -23 45 30 30
insert_0      3 6 4 6 8 4 2 9 6 4 6 3
insert_0      3 6 4 6 8 4 2 9 6 4 6 3
insert_end   -1 -5 12 3 45 -23 45 30 30
insert_end   -1 -5 12 3 45 -23 45 30 30
insert_0      3 6 4 6 8 4 2 9 6 4 6 3
insert_0      3 6 4 6 8 4 2 9 6 4 6 3
dim
Lista tem 84 itens

print_0
Lista(0)= 3

print_end
Lista(end)= 30

print
Lista= 3 6 4 6 9 2 4 8 6 4 6 3 3 6 4 6 9 2 4 8 6 4 6 3 3 6 4 6 9 2 4 8 6 4 6 3 3 6 4 6 9 2 4 8 6 4 6 3 -1 -5 12 3 45 -23
 45 30 30 -1 -5 12 3 45 -23 45 30 30 -1 -5 12 3 45 -23 45 30 30 -1 -5 12 3 45 -23 45 30 30

delete_pos 2
delete_pos 5
delete_pos 15
delete_pos 23
delete_end
delete_end
delete_end
delete_end
delete_end
delete_end
delete_end
delete_end
delete_end
delete_end
dim
Lista tem 70 itens

print
Lista= 3 6 6 9 2 8 6 4 6 3 3 6 4 6 9 4 8 6 4 6 3 3 6 6 9 2 4 8 6 4 6 3 3 6 4 6 9 2 4 8 6 4 6 3 -1 -5 12 3 45 -23 45 30 3
0 -1 -5 12 3 45 -23 45 30 30 -1 -5 12 3 45 -23 45 30

invert_range   0 30
print
Lista= 6 4 6 8 4 2 9 6 6 3 3 6 4 6 8 4 9 6 4 6 3 3 6 4 6 8 2 9 6 6 3 3 3 6 4 6 9 2 4 8 6 4 6 3 -1 -5 12 3 45 -23 45 30 3
0 -1 -5 12 3 45 -23 45 30 30 -1 -5 12 3 45 -23 45 30
```