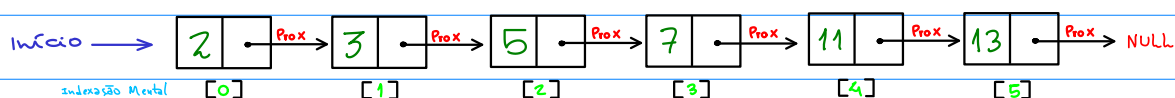


* Discente: Paulo Henrique Diniz de Lima Alencar. * Ciência da Computação
* Revisão - Estrutura de Dados.

1º) Descreva/ou desenhe passo a passo como funciona a remoção na n -ésima posição de uma lista encadeada com ponteiro apenas no início.

Resolução:



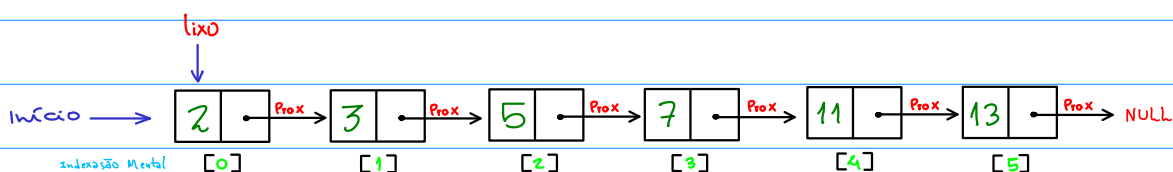
* tamanho da lista encadeada $\rightarrow 6$

* Remover o primeiro Nó da lista:

1º Passo: inicialmente precisamos criar um ponteiro cujo o identificador será lixo. Este ponteiro vai receber o endereço de memória do 1º Nó da lista encadeada, logo apontará para o 1º Nó.

Nossa lista agora:

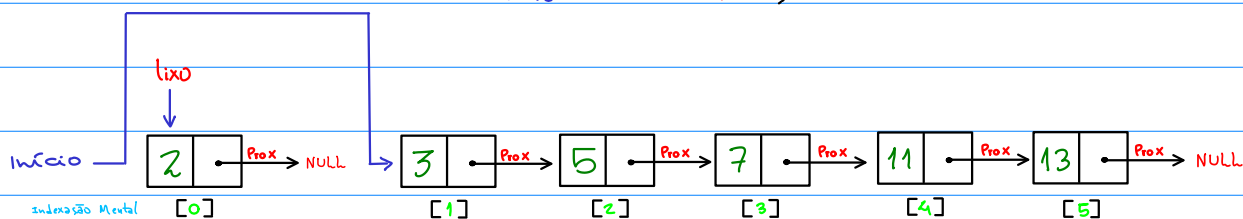
lixo = início;



2º Passo: agora é necessário atualizar o ponteiro início, pois como o 1º Nó será removido, o novo início de nossa lista será o Nó com o valor 3. Para fazer isso, basta escrever a seguinte operação:

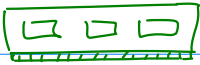
Nossa lista agora:

início = início \rightarrow prox;

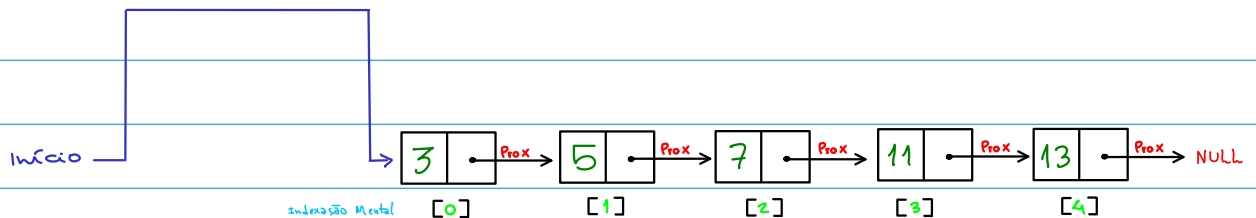


3º Passo: com **lixo** apontando para o **Nó** que será removido, podemos recuperar o **valor** presente no **Nó** para mostrar futuramente ao programador. A operação necessária é:

$\text{int removido} = \text{lixo} \rightarrow \text{valor};$

4º Passo: agora precisamos remover nosso 1º **Nó** definitivamente. Para fazer isso é preciso desalocar a porção de  (**memória**) ocupada pelo 1º **Nó**. Assim, `free(lixo)` vai informar ao S.O que o bloco de bytes apontado por **lixo**, está livre para reciclagem. Lembrando que também é importante reduzir em 1 unidade a variável que representa o tamanho da lista.

Nossa lista agora:



* tamanho da lista encadeada --> 5

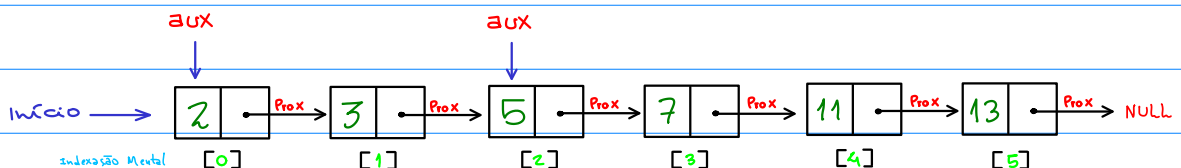
* Remoção no "meio-Fim":

1º passo: precisamos de uma variável especial, que vai percorrer **Nó** a **Nó** da lista encadeada, até um **Nó** antes da posição que desejamos remover. Essa variável terá como identificador a "palavra" **aux**.

Vamos supor que desejamos remover o elemento da posição [3]:

* **aux** começa aqui:

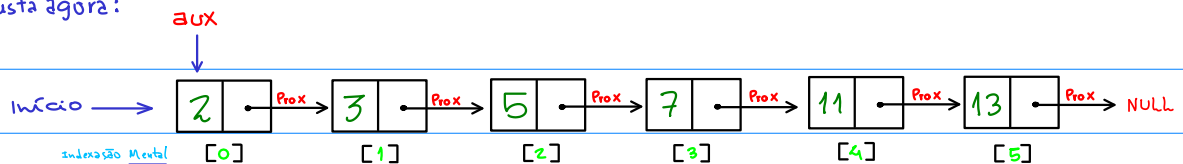
aux vai parar aqui:



Vamos precisar codificar o seguinte comando para que **aux** aponte para o 1º **Nó** da lista encadeada:

$\text{No}^* \text{aux} = \text{inicio};$

Nossa lista agora:



- 2º passo: precisamos que **aux** aponte para o Nó seguinte da lista encadeada, até chegar no **Nó** que valor **5**. Para isso, será necessário fazer uso de uma **estrutura de repetição**, mais especificamente o "for".

Atualizando nossos comandos:

⋮

NO* **aux** = início ;

int **i**;

for (**i** = 0 ; **i** < posicao - 1 ; **i**++) {

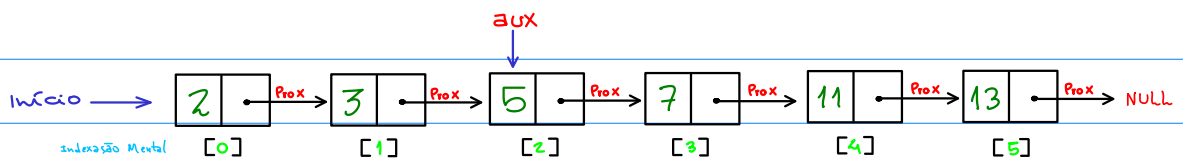
aux = **aux** → **prox** ;

// aponta para o **Nó** seguinte.

}

- 3º passo: agora vamos precisar de uma variável cujo o identificador será **lixo**. Ela será um ponteiro que vai apontar para o **Nó** que será removido, ou seja, o **Nó** que possui valor **7**.

Nossa lista agora:



NO* **lixo** ;

⋮

NO* **aux** = início ;

int **i**;

for (**i** = 0 ; **i** < posicao - 1 ; **i**++) {

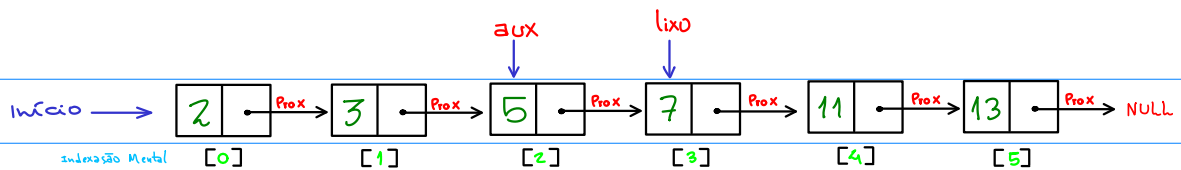
aux = **aux** → **prox** ;

}

lixo = **aux** → **prox** ;

// nova linha

Nossa lista agora:

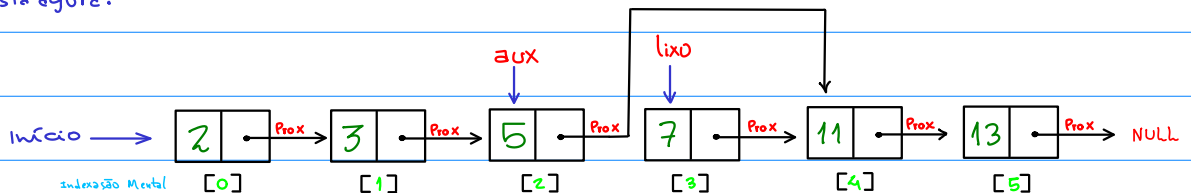


4º passo: após isso, como o Nó com valor 7 será removido, precisamos fazer com que $aux \rightarrow prox$ aponte para o Nó seguinte de nossa lista encadeada. Assim $aux \rightarrow prox$ precisa apontar para o Nó de valor 11.

Atualizando Nosso código:

```
NO* lixo;  
:  
NO* aux = inicio;  
int i;  
for (i = 0; i < posicao - 1; i++) {  
    aux = aux -> prox;  
}  
lixo = aux -> prox;  
aux -> prox = aux -> prox -> prox; // nova linha
```

Nossa lista agora:



5º passo: agora podemos desalocar  (memória) e também reduzir em 1 unidade a variável responsável por armazenar o tamanho de nossa lista encadeada, afinal retiramos um Nó.

Atualizando o código:

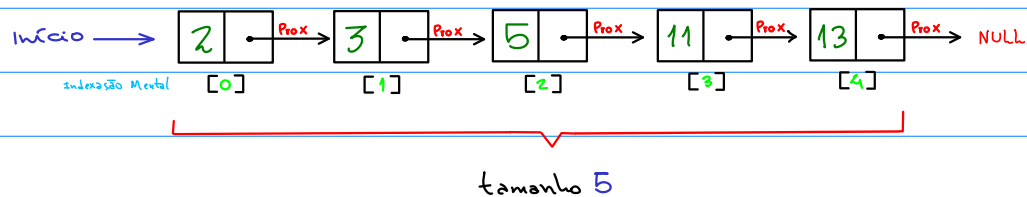
```

No* lixo;
⋮
No* aux = inicio;
int i;
for (i = 0; i < posicao - 1; i++) {
    aux = aux -> prox;
}
lixo = aux -> prox;
aux -> prox = aux -> prox -> prox;
free(lixo); // nova linha
tamanho--; // nova linha
⋮

```

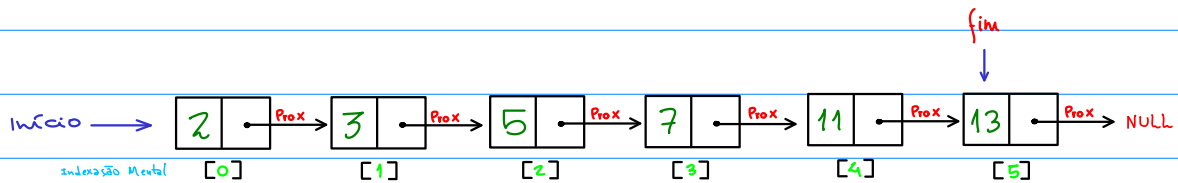
Essa função `free` ficou responsável por desalocar a porção de memória que foi alocada por `malloc`. Assim, `free(lixo)` vai informar ao S.O que o bloco de bytes apontado por `lixo`, está livre para reciclagem.

Nossa lista após remover o Nó com valor 7:



2º) Descreva e/ou desenhe passo a passo como funciona a adição na n-ésima posição de uma lista encadeada com ponteiro início e no fim. Resolução:

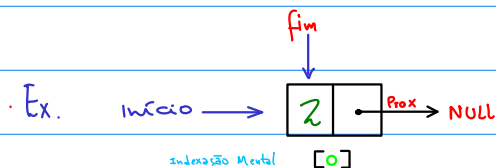
Exemplo de lista encadeada com ponteiro fim



A vantagem desse ponteiro fin, apontando para o último Nó da lista encadeada, é que não precisaremos mais percorrer as n posições da lista encadeada para adicionar um novo Nó no final da lista. Por consequência isso acaba tornando a operação de adicionar ao fin constante - $O(1)$.

* Lista encontra-se "vazia":

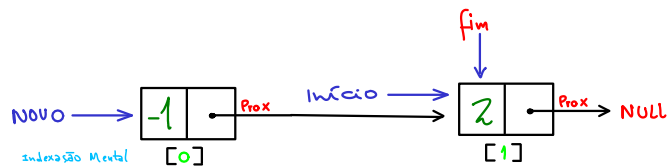
1º passo: precisamos fazer com que o ponteiro fin sempre aponte para o último Nó de nossa lista encadeada. Assim quando, adicionarmos o 1º Nó em nossa lista, precisamos fazer com que fin aponte para esse mesmo Nó.



```
...
if (início == NULL) {
    código:
    início = NOVO;
    fin = NOVO;
}
```

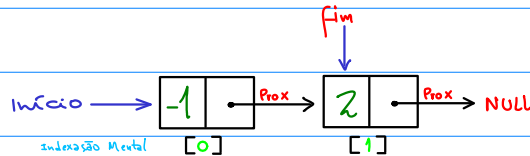
* Adicionar na posição 0, obs: lista já possui elementos

1º passo: precisamos fazer com que NOVO → prox detenha o endereço do 1º Nó. A operação necessária é a seguinte: NOVO → prox = início;



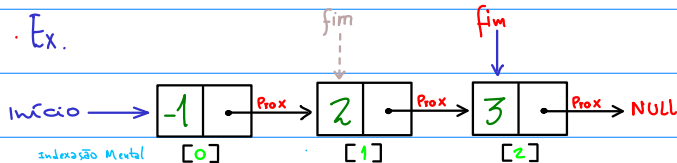
Agora precisamos atualizar nosso início, afinal o início da lista encadeada agora é outro. Operação necessária:

`início = NOVO;`



* Adicionando no Fim:

1º passo: Quando formos adicionar o novo Nó na próxima posição, precisamos atualizar nosso fim:



Ex.

```

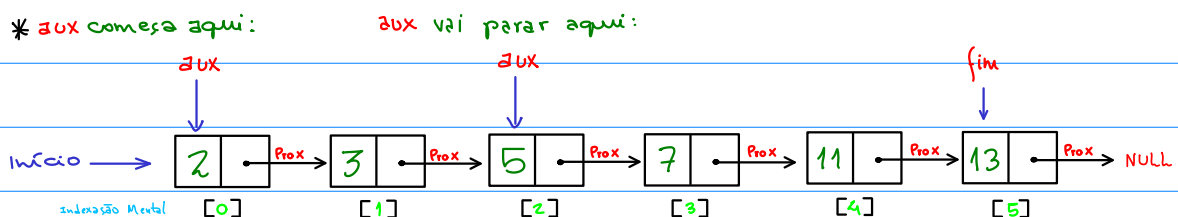
else if (posição == tamanho) {
    fim -> prox = Novo;
    fim = Novo;
}

```

* Adicionando no "Meio":

1º passo: para adicionar no meio, ainda será necessário fazer uso de uma estrutura de repetição em conjunto com o ponteiro especial nomeado aux, que vai percorrer Nó a Nó até a posição anterior que desejamos que desejamos adicionar.

Quero adicionar um Novo Nó na posição [3]



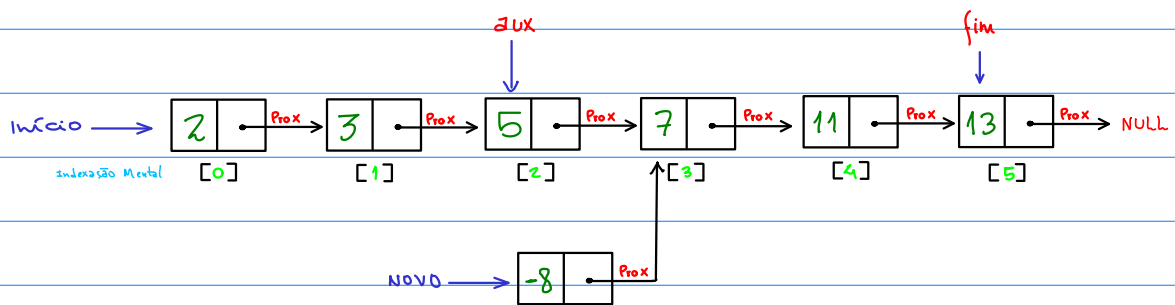
```

    ...
    No* aux = inicio;
    int i;
    for (i = 0; i < posicao - 1; i++) {
        aux = aux -> prox;
    }

```

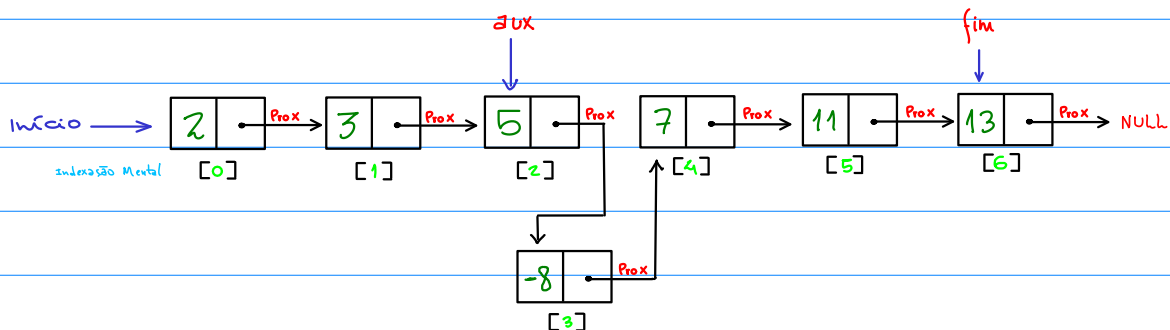
2º passo: agora precisamos que o Novo → prox aponte para o Nó que possui o valor 7.
A operação necessária é:

$\text{Novo} \rightarrow \text{prox} = \text{aux} \rightarrow \text{prox};$



3º passo: por fim, precisamos fazer que aux → prox aponte para o Nó Novo, que possui valor -8. A operação necessária:

$\text{aux} \rightarrow \text{prox} = \text{Novo};$



por fim, não podemos esquecer de adicionar 1 unidade na variável que representa o tamanho da lista.

3º) Descreva quais são as complexidades "simplificadas" (linear ou constante) dos casos adicionar no início, no fim e n-ésima posição de uma lista encadeada com ponteiro no início apenas e de uma lista encadeada com ponteiro no início e no fim. Justifique quando as complexidades forem diferentes para as duas listas.

Resolução:

* Apenas com ponteiro início:

- adicionar no início → constante $\boxed{=}$
- adicionar no fim → linear $\boxed{\neq}$
- adicionar no "meio" → linear $\boxed{=}$

* Com ponteiro início e fim:

- adicionar no início → constante
- adicionar no fim → constante
- adicionar no "meio" → linear

→ Para adicionar no fim de uma lista encadeada quando a lista possui o ponteiro fim sua complexidade será Constante. Sem fazer uso do ponteiro fim, teremos uma complexidade linear, para adicionarmos no fim da lista encadeada. Essa diferença ocorre pois, sem o ponteiro fim é necessário fazer uso de uma variável aux para percorrer todas as posições da lista. Isso torna a complexidade linear, afinal quanto maior o valor de n elementos em minha lista encadeada, maior será o número de operações.

Referências:

FERNANDES, Tatiane et al. Estrutura de dados - Uma abordagem gráfica do Laboratório de Estruturas de dados. 1ed. Ceará. 2020.