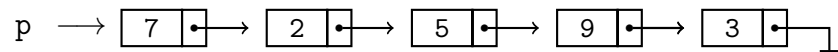


Estruturas de Dados

aula 06: Manipulando listas encadeadas

1 Introdução

Na aula passada, nós vimos as listas encadeadas



E vimos algoritmos que percorriam a lista encadeada para inspecionar os seus elementos.

Hoje nós vamos aprender a fazer modificações na lista.

E a tarefa mais simples de todas é inserir um novo elemento na lista.

Mas para isso, é preciso aprender a criar um novo registro.

Só que isso é uma coisa bem fácil

Quer dizer, basta utilizar o comando `Novo()`, que nos dá um ponteiro para um registro vazio



Daí, nós podemos preencher o registro como nós quisermos

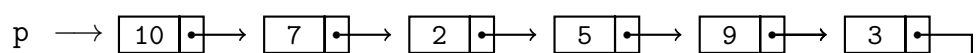


Legal.

Isso já é suficiente para inserir um novo registro na lista do exemplo acima

```
q = Novo(RegInt)
q->num = 10
q->prox = p           // aponta para o 1o elemento
p = q                // esse é o novo 1o elemento
```

O que nos dá



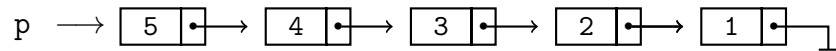
Agora, a gente também pode criar uma lista de qualquer tamanho a partir do zero

```

p = Nulo
Para k=1 Ate 5
    q = Novo (RegInt)
    q->num = k
    q->prox = p
    p = q

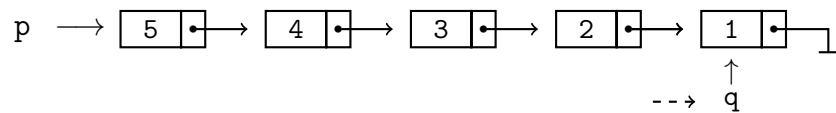
```

O que produz a lista



E agora, a gente também pode escrever uma função que insere um novo elemento no fim da lista.

Para isso, a gente leva um ponteiro auxiliar até o último elemento



E daí, a gente realiza a operação de inserção

```

func insercaoFim (k, p)

    Se ( p == Nulo )                                // a lista ainda está vazia
        r = Novo (RegInt)
        r->num = k
        r->prox = Nulo
        Retorna (r)

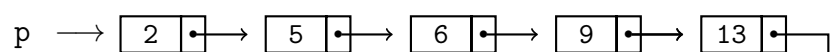
    q = p                                           //
    Enquanto ( q->prox != Nulo )                   // vai até o último elemento
        q = q->prox                                //

    r = Novo (RegInt)
    r->num = k                                       //
    r->prox = Nulo                                 // insere o registro no fim
    q->prox = r                                     //
    Retorna (p)

```

2 Manipulando a lista encadeada

Agora imagine que nós temos uma lista encadeada ordenada



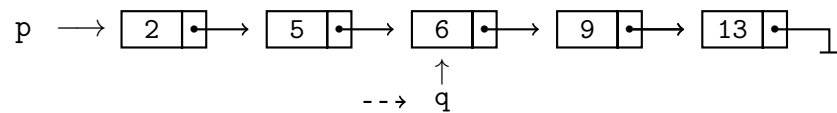
E imagine que a gente quer inserir um novo elemento k na lista.

Como é que a gente faz isso?

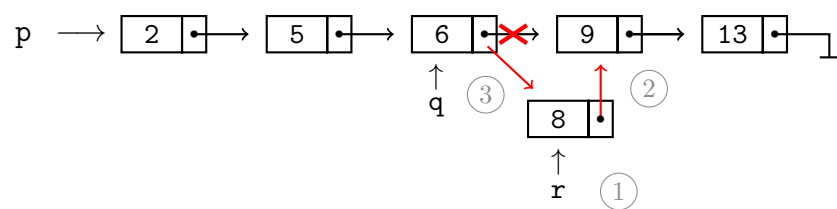
Bom, é quase a mesma coisa.

Quer dizer, imagine que a gente quer inserir o 8 na lista acima.

Então, a gente vai até o último elemento que ainda é menor que o 8



E daí, a gente faz a inserção assim



— (*you saw that the order in which the things are done is important?*)

Abaixo nós temos o código da função que implementa a inserção na lista ordenada

```
func insercaoOrd (k, p)

    Se ( p == Nulo OU p->num > k)                                // inserção na primeira posição
        r = Novo (RegInt)
        r->num = k
        r->prox = Nulo
        Retorna (r)

    q = p                                                         //
    Enquanto (q->prox != Nulo E (q->prox)->num < k)              // vai até o último menor que k
        q = q->prox                                              //

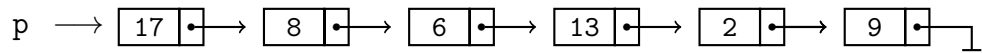
    r = Novo (RegInt)
    r->num = k                                                     //
    r->prox = q->prox                                              // insere o registro no fim
    q->prox = r                                                    //
    Retorna (p)
```

A seguir, nós vamos ver mais exemplos.

Exemplos

a. Movendo o primeiro para o fim

Imagine que o ponteiro p aponta para uma lista encadeada

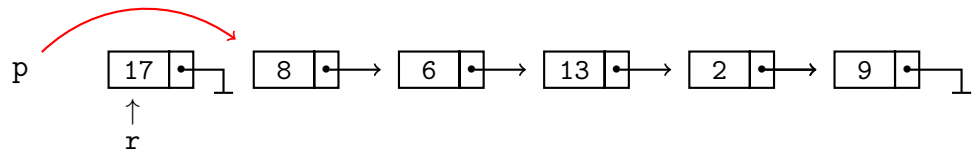


A tarefa é

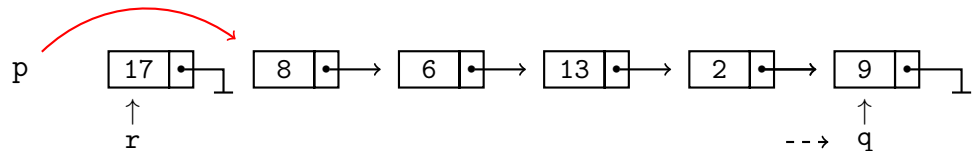
→ *Mover o primeiro elemento para o final da lista*

Bom, aqui a gente vai fazer a coisa em 3 etapas

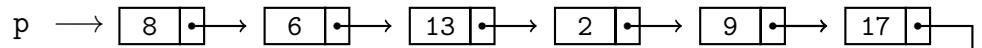
- primeiro a gente remove o primeiro elemento da lista



- daí a gente leva um ponteiro auxiliar até o último elemento



- e daí a gente faz a inserção no fim



A coisa fica assim

```
func primeiroFim (p)

    Se ( p == Nulo )          Retorna (p)           // lista vazia
    Se ( p->prox == Nulo )    Retorna (p)           // o 1o já é o último

    r = p                     //
    p = p->prox                // 1a Etapa
    r->prox = Nulo             //

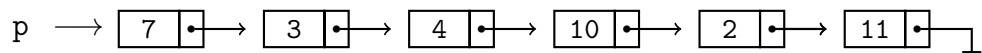
    q = p                     //
    Enquanto (q->prox != Nulo ) // 2a Etapa
        q = q->prox           //

    q->prox = r                // 3a Etapa
    Retorna (p)                //
```

◇

b. Removendo o elemento k

Imagine que o ponteiro p aponta para uma lista encadeada

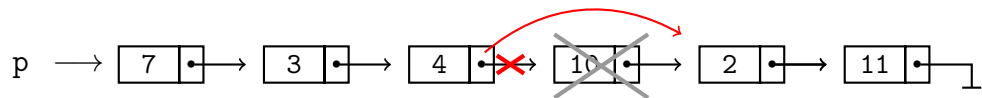


A tarefa é

→ *Remover o elemento k da lista*

Bom, aqui a gente precisa ter algum cuidado.

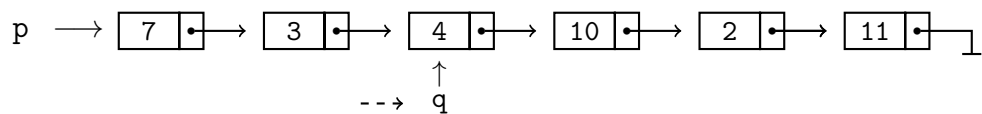
Abaixo nós temos o que precisa ser feito para remover o 10 da lista



Quer dizer, para remover o 10 é preciso modificar o registro do elemento que vem antes do 10.

E para modificar esse registro é preciso ter um ponteiro apontado para ele.

Então, para fazer a remoção do elemento k, a gente precisa levar um ponteiro auxiliar até o registro que fica antes do k



E daí, a gente realiza a operação de remoção.

A coisa fica assim

```
func opRemoção ( k, p )

    Se ( p == Nulo )      Retorna (p)           // o k não está lá ...

    Se ( p->num == k )    // o k é o primeiro elemento
        r = p             // segura ...
        p = p->prox        //
        free (r)           // e libera o registro do k
        Retorna (p)       //

    q = p;   achei = 0

    Enquanto (q->prox != Nulo )           // procura o k
        Se ( q->num == k )
            achei = 1;   Quebra
            q = q->prox

    Se ( achei == 0 )           // o k não está lá ...
        Retorna (p)
```

```

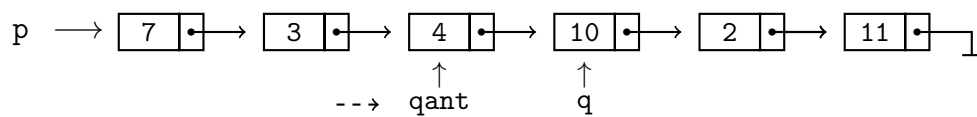
Senão
    r = p->prox                                // segura ...
    q->prox = r->prox
    free (r)                                    // e libera o registro do k
    Retorna (p)

```

Legal.

Mas, essa não é a única maneira de fazer as coisas.

Quer dizer, a gente também pode percorrer a lista com dois ponteiros, para chegar até o *k* e o elemento que fica atrás dele



E agora, é fácil fazer a remoção.

A coisa fica assim

```

func opRemoção2 ( k, p )

    Se ( p == Nulo )      Retorna (p)                // lista vazia

    q = p;    qant = Nulo

    Enquanto (q->prox != Nulo )                    // procurando o k
        Se ( q->num == k )
            achei = 1;    Quebra
        q = q->prox

    Se ( q == Nulo )      Retorna (p)                // o k não está lá ...

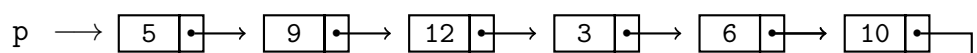
    Senão
        Se ( qant == Nulo )    p = q->prox          // o k é o 1o elemento
        Senão                  qant->prox = q->prox
        free (q)
        Retorna (p)

```

◇

c. Movendo o maior para o fim

Imagine que o ponteiro *p* aponta para uma lista encadeada



A tarefa é

→ *Mover o maior elemento para o fim da lista*

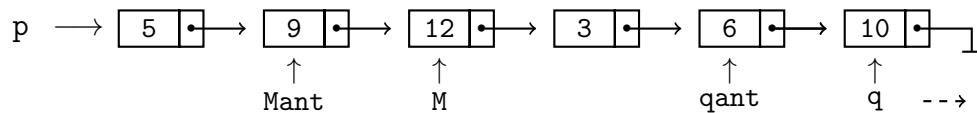
Bom, aqui a gente precisa fazer a coisa em etapas

1. primeiro, a gente encontra o maior elemento
2. daí, a gente remove esse elemento da lista
3. e daí, o maior elemento é recolocado no final da lista

Certo.

Para encontrar o maior, a gente vai utilizar a técnica de percorrer a lista com dois ponteiros.

E daí, sempre que o ponteiro para o maior é atualizado, nós também atualizamos o ponteiro que aponta para o elemento que fica antes do maior.



A coisa fica assim

```
func maiorFim ( p )

Se ( p == Nulo OU p->prox == Nulo)   Retorna (p)

M = p;   Mant = Nulo                                     // o maior até aqui

q = p->prox;   qant = p

Enquanto (q != Nulo )                                // procurando o maior
    Se ( q->num > M->num )
        M = q;   Mant = qant
        qant = q;   q = q->prox

Se ( M->prox == Nulo )   Retorna (p)                   // o maior já é o último

Se ( Mant == Nulo )   p = M->prox                       // o maior é o 1o elemento
Senão
    Mant->prox = M->prox
M->prox = Nulo

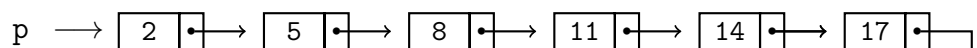
qant->prox = M                                           // 3a etapa
Retorna (p)
```

Nota: Depois que a gente termina de percorrer a lista (1a etapa), o ponteiro `qant` fica posicionado no último elemento. E é por isso que a 3a etapa fica tão simples.

◇

d. A operação de atualização

Imagine que o ponteiro `p` aponta para uma lista encadeada ordenada



A tarefa é

- encontrar o elemento x
- modificar o seu valor para y*
- e colocá-lo no seu lugar correto na lista ordenada*

Aqui mais uma vez, a gente precisa fazer a coisa em etapas

1. encontrar o elemento x
2. atualizar o seu valor e remove-lo da lista
3. colocar o registro no lugar correto

Para encontrar e remover o elemento, a gente vai utilizar a técnica de percorrer a lista com dois ponteiros.

Daí, se $y > x$, a gente continua percorrendo a lista para encontrar o novo lugar do registro.

E se $y < x$, a gente precisa percorrer a lista desde o início outra vez.

A coisa fica assim

```
func opAtualização ( x, y, p )

    q = p;    qant = Nulo;    achei = 0                //
    Enquanto ( q != Nulo )                               //
        Se ( q->num = x )                                // 1a Etapa
            achei = 1;    Quebra                          //
            qant = q;    q = q->prox                      //

    Se ( achei == 0 )    Retorna (p)

    Se ( qant == Nulo )    p = p->prox                    //
    Senão                  qant->prox = q->prox           // 2a Etapa
    q->num = y                //

    Se ( y > x )                //
        Enquanto ( qant->prox != Nulo)                    //
            Se ( (qant->prox)->num > y )                    // 3a Etapa (a)
                Quebra                                      //
                qant = qant->prox                          //
            q->prox = qant->prox                            //
            qant->prox = q                                  //

    Senão                //
        Se ( p->num > y )                                    //
            q->prox = p                                      // 3a etapa (b)
            p = q                                            //
```

```

Senão
    qant = p
    Enquanto ( qant->prox != Nulo)
        Se ( (qant->prox)->num > y )
            Quebra
            qant = qant->prox
        q->prox = qant->prox
        qant->prox = q

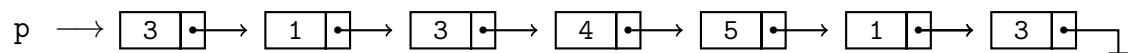
Retorna (p)

```

◇

e. Todas as cópias do primeiro no início

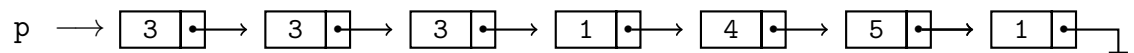
Imagine que o ponteiro p aponta para uma lista encadeada que contém elementos repetidos



A tarefa é

→ *Mover todas as cópias do primeiro elemento para o início da lista*

No exemplo acima, isso nos daria



Essa tarefa não é realmente difícil.

É só percorrer a lista com dois ponteiros.

E daí, a cada vez que uma cópia do primeiro elemento é encontrada, ela é removida da lista e colocada na primeira posição.

A coisa fica assim

```

func CPI ( p )                                     // Cópias do Primeiro no Início

    Se ( p == Nulo OU p->prox == Nulo)    Retorna (p)

    q = p->prox;    qant = p
    Enquanto (q != Nulo )
        Se ( q->num == p->num )                // Você é cópia do primeiro?
            qant->prox = q->prox;                // remove da lista
            q->prox = p;                        // reinsere ...
            p = q;                             // no início
            q = qant->prox;

    Senão
        qant = q;    q = q->prox

    Retorna (p)

```

◇