Listas

Sumário:
O TAD Lista.
Implementação de listas.
Listas ligadas. Listas duplamente ligadas

25-Março-21
1-Abril-21

Listas

- Uma lista é uma colecção de elementos indexada
  - posso aceder a elementos da lista dando a posição do elemento que quero inserir/remover/consultar
  - Essa posição, não está relacionada com a ordem pela qual inserimos o elemento na lista
    - não há protocolos de acesso (LIFO; FIFO); é indexada
- Seja $L \equiv a_1, a_2, a_3, \ldots, a_n$, uma lista
  - Trata-se duma lista com n elementos; (lista de tamanho n)

O TAD Lista

- $L \equiv a_1, a_2, a_3, \ldots, a_n$
  - O primeiro elemento da lista é a_1
  - O último elemento da lista é a_n
  - Uma lista vazia não tem elementos
  - O elemento a_{i+1} sucede o elemento a_i $(\forall i < n)$
  - O elemento a_{i-1} antecede o elemento a_i $(\forall i > 1)$
  - A posição na lista do elemento a_i é i

Listas

- Associadas a estas definições está um conjunto de operações que é usual incluirmos no TAD Lista:
  - tornar a lista vazia
  - listar os seus elementos
  - pesquisa (find); retorna a posição correspondente à primeira ocorrência duma chave na lista
  - inserir (insert): insere uma chave numa determinada posição na lista
  - remover (remove): remove a primeira ocorrência duma chave; remove o elemento numa determinada posição

Listas

- Exemplo: (15;34;9;18;93;6)
  - find(18) 4
  - insert(10,3) (15;34;10;9;18;93;6)
  - remove(9) (15;34;10;18;93;6)

Implementação(Array)

- Implementamos Stacks, Queues, com arrays e bem(complexidade!)

- Todas as operações que referimos, implementam-se facilmente com arrays

  - Problemas?

    - alocação prévia de espaço é uma limitação, e dimensão das listas, sobrestimar a dimensão…

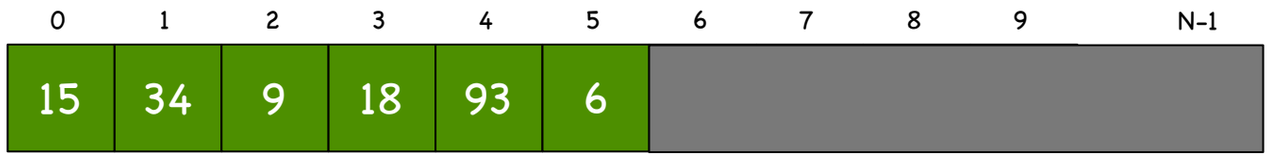
- Mas…também haverá vantagens

  - listagens e pesquisa são operações que podem ser realizadas em tempo linear (??)

  - A operação findIth é realizada em tempo constante

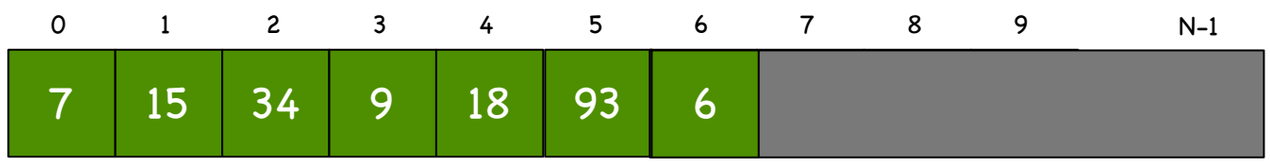
Implementação com array

- Analisemos a inserção:
  - inserir(7,P(0))



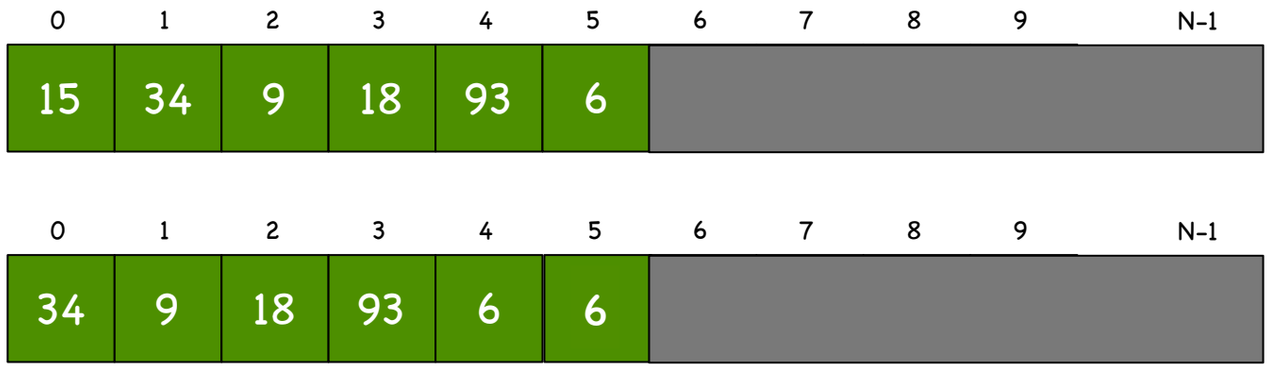
  - Toda a lista tem de ser deslocada uma posição para a frente

  - e só depois posso introduzir o 7



Implementação (Array)

- Para remover o primeiro elemento toda a lista tem de ser deslocada uma posição para trás.



- São o pior dos casos (inserção/remoção) e é $\Theta(n)$, para n o nº de elementos na lista

Implementação (Array)

- No caso médio, metade da lista tem de ser deslocada ($O(n)$)
- Construir uma lista inserindo n elementos requer tempo quadrático
- Por o tempo requerido para inserir e remover ser tão penalizante e por ter de ser conhecida antecipadamente a dimensão das listas, o uso de arrays na implementação de listas preterido por outras técnicas

Listas Ligadas

- O modo de evitar o tempo linear nas inserções e remoções é não exigir que os elementos da lista sejam guardados contiguamente em memória.

- É a ideia das listas ligadas.

- Uma lista ligada é constituída por uma série de nós, que não são necessariamente adjacentes em memória.

- Cada nó contem além do elemento, um apontador para nó que o sucede na lista. Este apontador designa-se usualmente por “next”

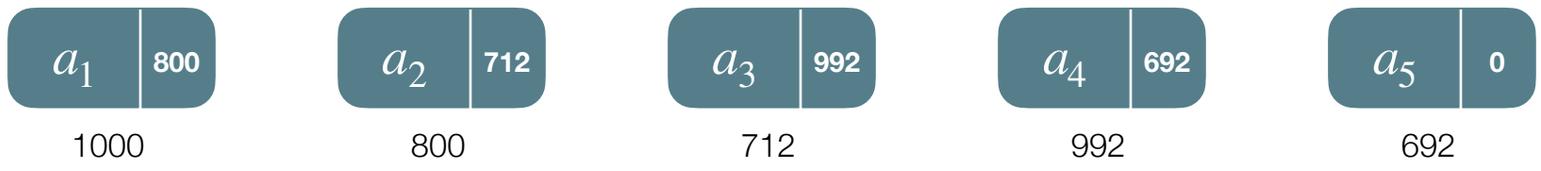
- Para o último nó da lista, este apontador é NULL

Listas ligadas

- Recorde-se:
  - umm apontador é uma variável que contem o endereço de memória, que conterá em si outra informação armazenada
- Se p for declarado como sendo um apontador para uma estrutura, então
  - p é encarado como a localização na memória principal, onde essa estrutura pode ser encontrada.
  - O campo duma estrutura pode ser acedido através de p->nome_campo

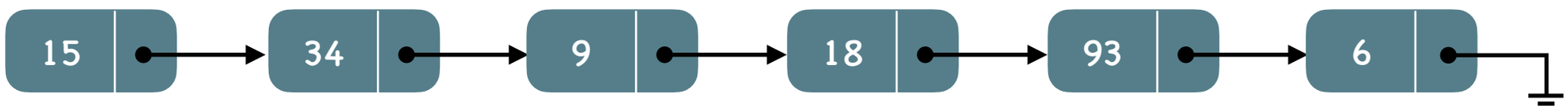
Listas Ligadas

- A lista $(a_1, a_2, a_3, a_4, a_5)$ pode ser visualizada como



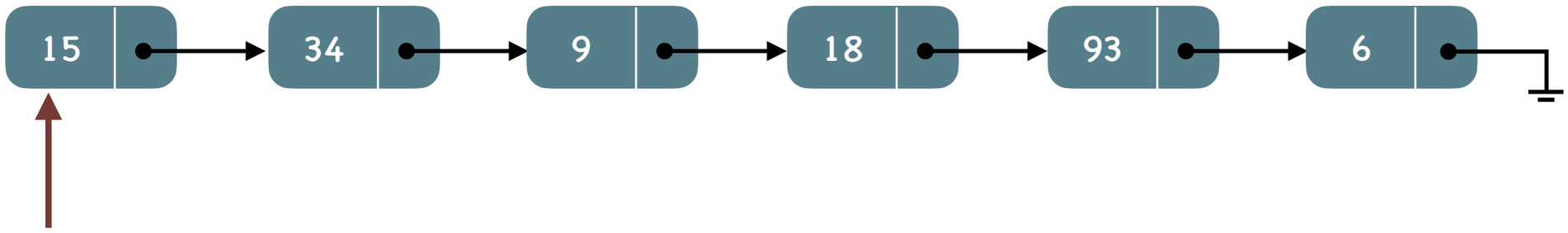
- Mas ilustraremos os apontadores com setas (mais intuitivo). A mesma lista

- Ou a lista (15;34;9;18;93;6)



Operações sobre listas ligadas

- Para implementar Print_List(L), ou Find(key,L), é só passar um apontador para o primeiro nó da lista, e percorrê-la usando os apontadores next



- Trata-se claramente duma operação de complexidade linear em ambos os casos, e numa implementação com um array igualmente.

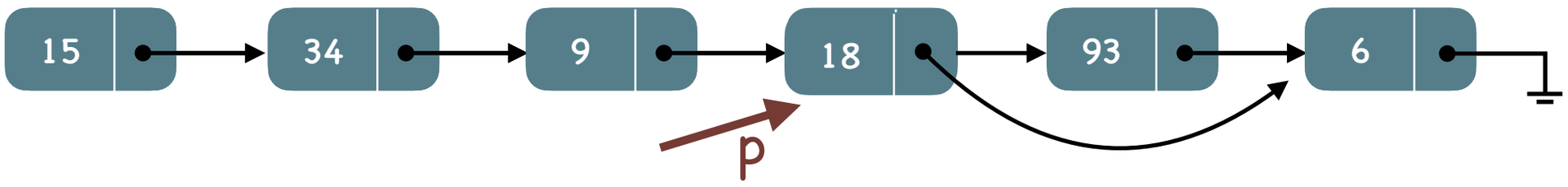
- Na operação Find_kth(L,i), ou seja obter o elemento na i-ésima posição da lista, agora (LL), ficamos a perder

Listas Ligadas

- As operações sobre listas ligadas, passam, regra geral, por estabelecer novas ligações, ou seja, manipular os apontadores

- A remoção de um nó da lista corresponde a fazer um by-pass

- Exemplo: Delete(93)



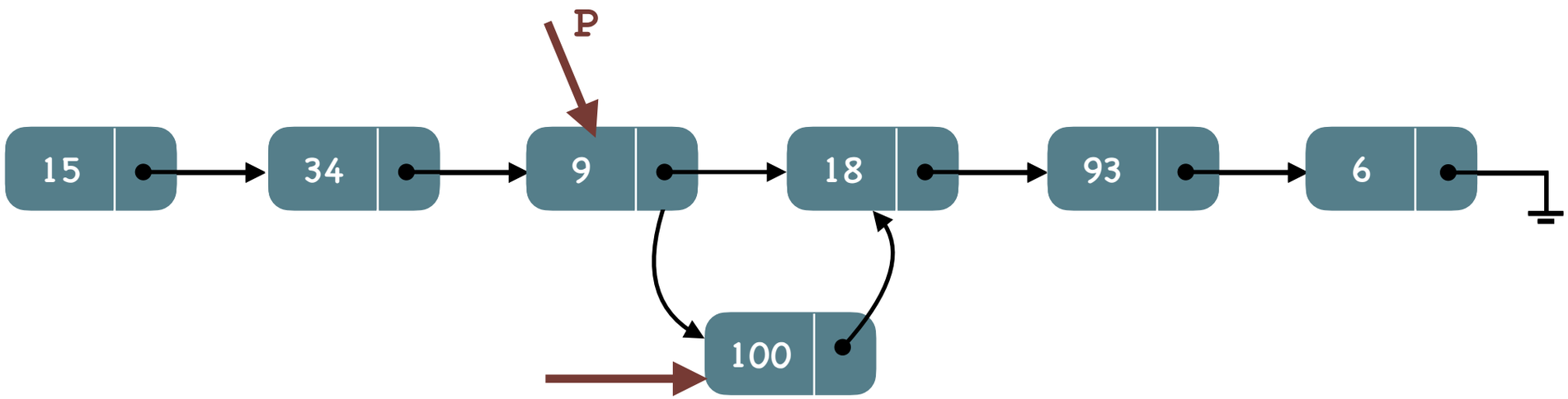
- É fácil, basta actualizar um apontador, mas

  - Não é no nó que queremos remover…

Listas Ligadas

- A inserção de um elemento na lista, passa pela criação dum nó com o novo elemento e pelo estabelecimento correcto das novas ligações

  - Exemplo: insere(100,4)




```
Novo_no->Element=X;
Novo_no->Next=P->Next;   //esta
P->Next=Novo_no;         // e esta
```

Novo_no?
P? na figura?

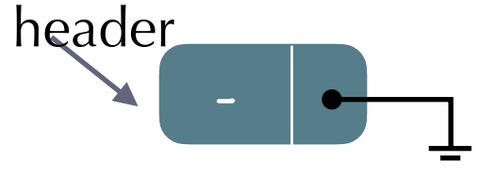
A ordem das operações é relevante

Implementação

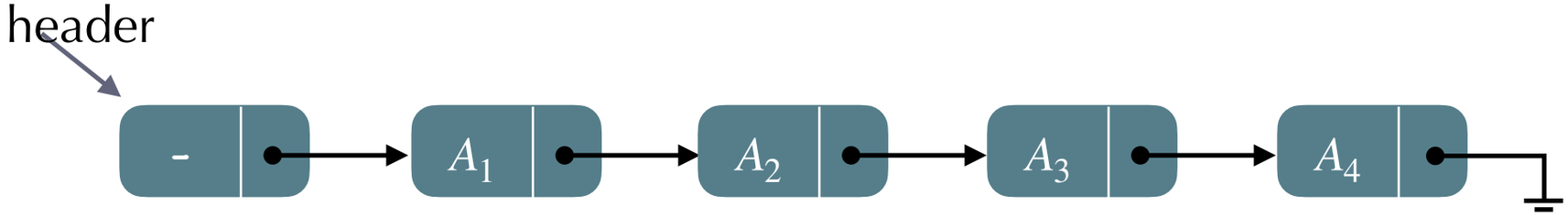
- As considerações apresentadas são suficientes para realizar uma implementação de listas ligadas. Mas:
  - Não é muito claro como realizar inserções no início da lista
  - Remover o elemento que está à cabeça da lista é um caso especial, pois modifica o início da lista
    - A remoção dum elemento requer procurar o nó que precede o elemento a remover e não há nó que anteceda o primeiro nó da lista
    - Por estes motivos é usual usar um nó sentinela, usualmente chamado de “header” ou ”dummy node”

Implementação

- Uma lista vazia:



- A lista $(A_1; A_2; A_3; A_4)$



Definição de Listas

```
struct Node
{
    ElementType Element;
    Position    Next;
};
```

```
#include <stdbool.h>
typedef int ElementType;

#ifndef List_h
#define List_h


struct Node;
typedef struct Node *PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;
```

```
List CreateList( List L ); //with dummy node
bool IsEmpty( List L );
bool IsLast( Position P, List L );
Position Find( ElementType X, List L );
void Delete( ElementType X, List L );
Position FindPrevious( ElementType X, List L );
void Insert( ElementType X, List L, Position P );
void DeleteList( List L );
Position Header( List L );
Position First( List L );
Position Advance( Position P );
ElementType Retrieve( Position P );
void PrintList(List L);

#endif    /* _List_H */
```

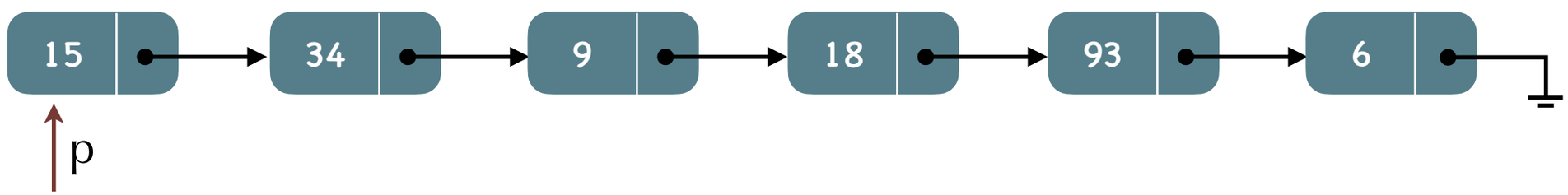
```c
int main(int argc, const char * argv[]) {
    List L=CreateList(NULL);

    for(int i=1;i<=10;i++){
        Insert(i, L, Header(L));
    }

    if (IsEmpty(L))
        printf("L is empty\n");
    else
        printf("Nope");

    PrintList(L);
    Position Fim=Header(L);
    printf("Começa em %d\n",Retrieve(Fim));
    while (!IsLast(Fim, L)){
        Fim=Advance(Fim);
    }
    //Fim pointer para a última posição
    printf("Último é %d \n",Retrieve(Fim));
    Insert(20,L,Fim);
    Delete(5,L);
    PrintList(L);
}
```

```
Nope
[10 ; 9 ; 8 ; 7 ; 6 ; 5 ; 4 ; 3 ; 2 ; 1 ]
Começa em 0
Último é 1
[10 ; 9 ; 8 ; 7 ; 6 ; 4 ; 3 ; 2 ; 1 ; 20 ]
Program ended with exit code: 0
```

LISTAS DUPLAMENTE LIGADAS

- Podemos reduzir a complexidade das operações se pudermos, além de andar para a frente, andarmos para trás nas listas?

- Como andamos (para a frente) numa lista?



- Seguimos os apontadores next dos nós

LISTAS DUPLAMENTE LIGADAS

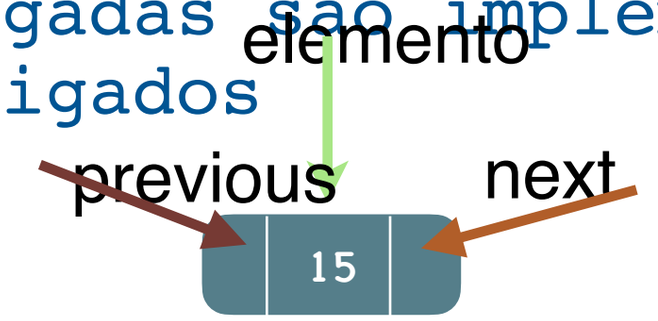
- Dada uma qualquer posição da lista, é possível, percorrer a lista que se inicia nessa posição, no sentido dos apontadores "next"

```
Position P= Header(L);
while (P!=NULL){
     P=Advance(P);
}
```

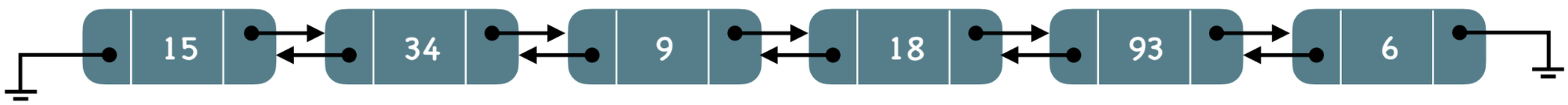
- Para percorrer uma lista no sentido inverso, esta implementação não ajuda, mas podemos criar outro tipo de nós, que além dum apontador para o nó seguinte, contenham um apontador para o nó anterior
  - É a ideia das listas duplamente ligadas

LISTAS DUPLAMENTE LIGADAS

- As listas duplamente ligadas são implementadas usando nós duplamente ligados



- As listas agora têm a representação

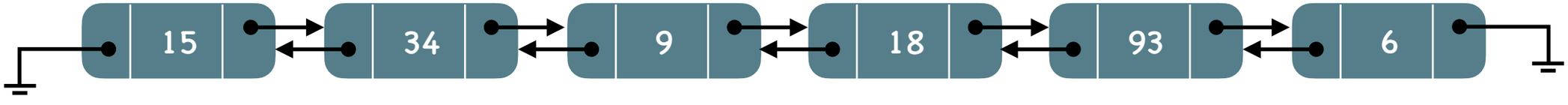


  - lista duplamente ligada sem “header”

- Todas as operações definidas para as listas ligadas são facilmente implementadas com as listas duplamente ligadas, mas há custos:

  - mais espaço ocupado

LISTAS DUPLAMENTE LIGADAS



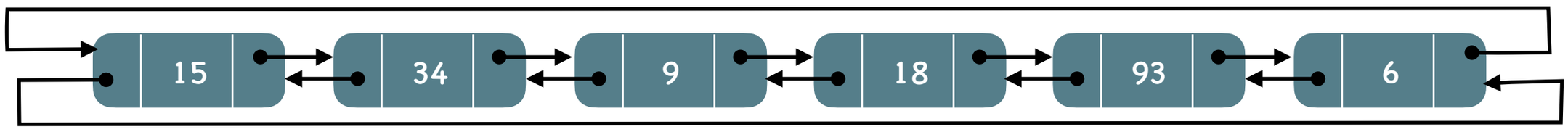
- Todas as operações definidas para as listas ligadas são facilmente implementadas com as listas duplamente ligadas, mas há custos:
  - mais espaço ocupado
  - mais ligações a corrigir, aquando das inserções ou remoções
  - "Não há bela sem senão"
    - Mais eficiente a remoção, dado agora que eu sei qual o nó anterior, à chave que quero remover

Listas Ligadas Circulares

- Outra convenção sobre as listas ligadas é que o apontador next do último nó aponte para o primeiro nó da lista

  - pode ser feito, em listas simplesmente ligadas ou duplamente ligadas, com header ou não. No caso de haver header, o último nó aponta para o header

- Serão mais "meticulosos" alguns testes, mas inserir no fim da lista pode ser facilitado.



Tarefas

- A implementação é para fazer(casa, aula prática…)
- Implementar Stacks com Listas Ligadas
- Implementar Queues com Listas ligadas