Apontamentos de Estruturas de Dados

Estruturas Ligadas

Ricardo Santos| rjs@estg.ipp.pt

Escola Superior de Tecnologia e Gestão
Instituto Politécnico do Porto

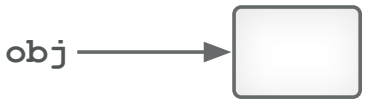
Última versão: Outubro de 2022

Índice

5. Estruturas Ligadas

Uma estrutura ligada ou lista ligada é uma estrutura de dados linear. Estas listas são sequências de objetos de uma classe autorreferenciada denominados de nó, conectados por *links* de referências - daí o termo estrutura ligada. Normalmente, um programa acede a uma lista ligada através de uma referência ao primeiro nó da lista. Para acedermos aos nós seguintes é usada a referência do *link* armazenado no nó anterior. Por convenção, a referência do *link* no último nó é definida como nula (`null`) para marcar o final da lista. Os dados são armazenados numa lista ligada dinamicamente, sendo os nós criados à medida que são necessários. Um nó pode conter dados de qualquer tipo incluindo referências a objetos de outras classes. Este tipo de estrutura pode por sua vez ser usado para implementar filas ou pilhas por exemplo.

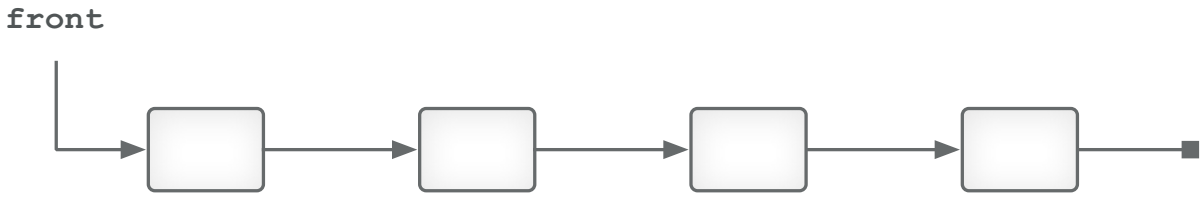
Uma estrutura ligada usa referências de objetos para ligar um objeto a outro. Lembre-se que uma variável com a referência de um objeto armazena o endereço desse mesmo objeto. Nesse sentido, uma referência de objeto pode ser vista como um apontador para objeto.



Por exemplo, o objeto `Person` pode conter uma variável com a referência para um outro objeto `Person`.

```
1. public class Person {
2.     private String name;
3.     private String address;
4.     private Person next; // a link to another Person object
5.     // whatever else
6. }
```

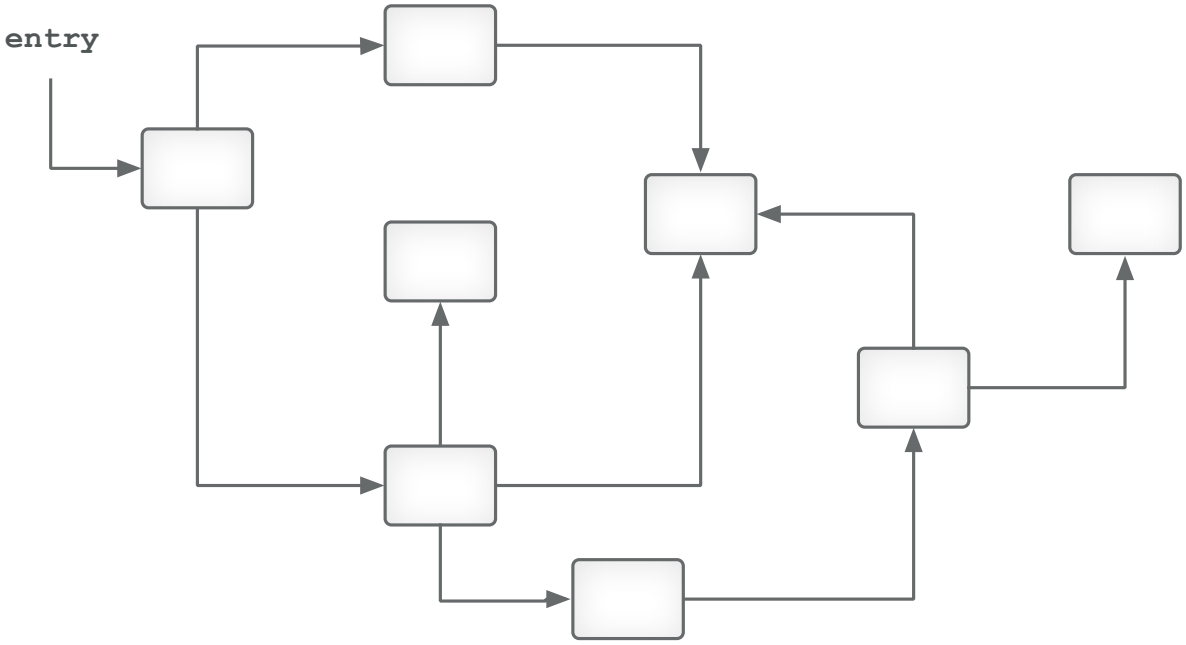
Este tipo de referência pode ser usada para formar uma lista ligada, na qual um objecto se refere ao seguinte, que remete para o próximo, etc. Cada objecto numa lista é genericamente denominado de nó. Uma lista ligada é uma estrutura de dados dinâmica em que seu tamanho aumenta e diminui conforme a necessidade, ao contrário de um *array*, cujo tamanho é fixo. Os objetos Java são criados dinamicamente quando são instanciados!



As listas de dados (estrutura de dados linear) podem ser armazenadas tanto em *arrays* como em listas ligadas, no entanto as listas ligadas oferecem várias vantagens. Uma lista ligada é apropriada quando o número de elementos de dados a serem armazenados na estrutura de dados é imprevisível. As listas ligadas como já foi referido são dinâmicas, portanto, o comprimento de uma lista pode aumentar ou diminuir conforme necessário. O tamanho de um *array* Java "convencional", como sabem, não pode ser alterado pois o tamanho do *array* é fixo no momento em que é criado. *Arrays* "convencionais" podem ficar cheios. As listas ligadas apenas ficam cheias quando o sistema tem memória insuficiente para atender às solicitações de alocação de armazenamento dinâmico. O *package* `java.util` por exemplo, contém a classe `LinkedList` para implementação e manipulação de listas ligadas que aumentam e diminuem durante a execução do programa.

Um *array* também pode ser declarado de forma a armazenar mais elementos do que o número de elementos esperados, mas faz com que seja desperdiçada memória. As listas ligadas oferecem uma melhor gestão da memória nessas situações já que permite que sejam adaptadas às necessidades de armazenamento em tempo de execução.

As referências de objetos permitem também criar estruturas não-lineares, como as árvores e os grafos.

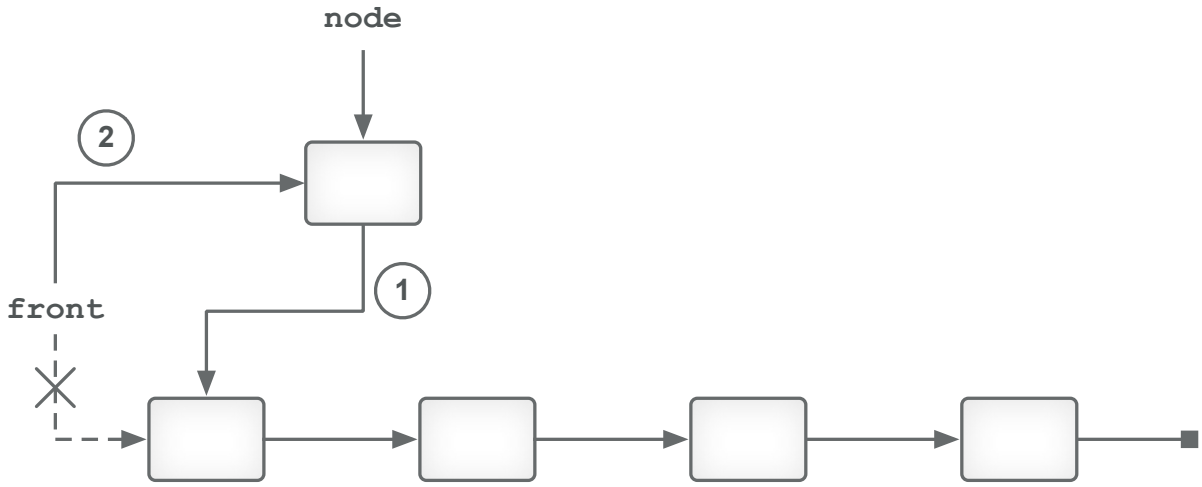


As referências numa lista ligada devem ser cuidadosamente geridas para manter a integridade da estrutura. Devem-se ter cuidados especiais para garantir que o ponto de entrada na lista é mantido adequadamente. A ordem em que certos passos são tomados é de extrema importância!

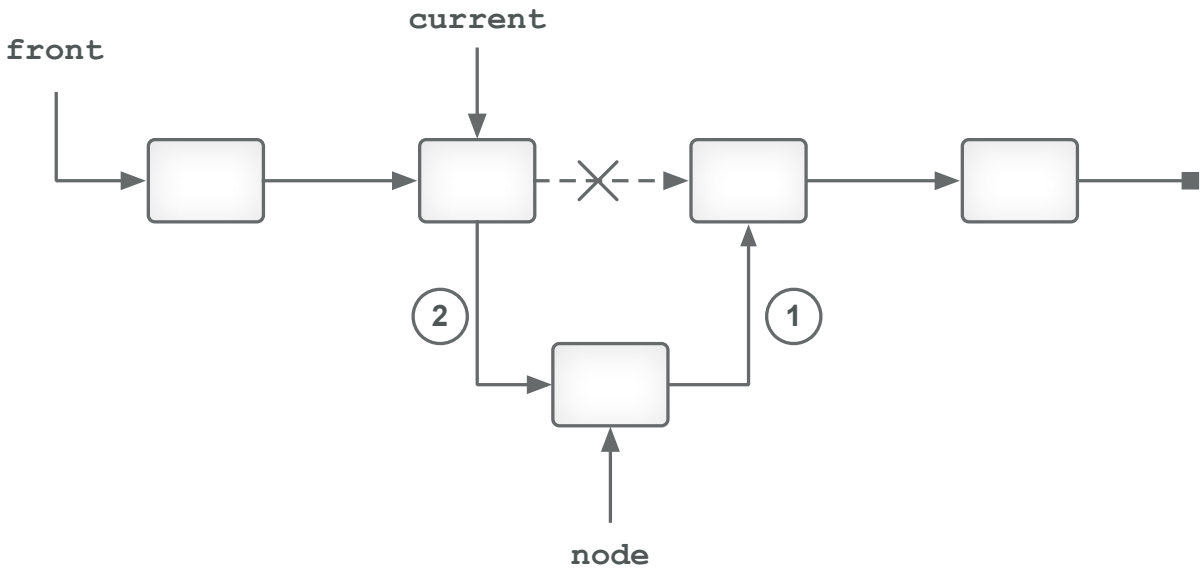
5.1. Listas Simplesmente Ligadas

Inserir um nó na lista ligada

Para inserirmos um nós numa lista temos essencialmente duas situações distintas: inserir um nó na cabeça (`front`) ou no meio da lista ligada.



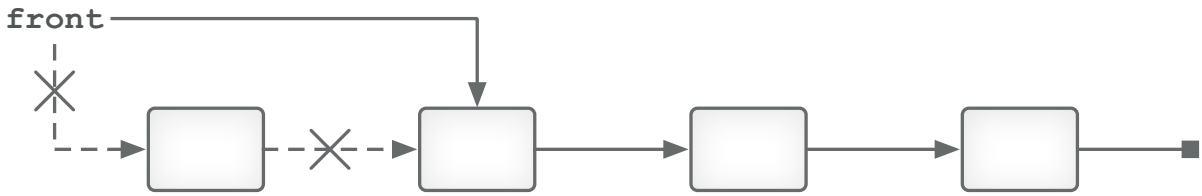
```
1. insertFirst(node)
2.     node.next = front
3.     Front = node
```



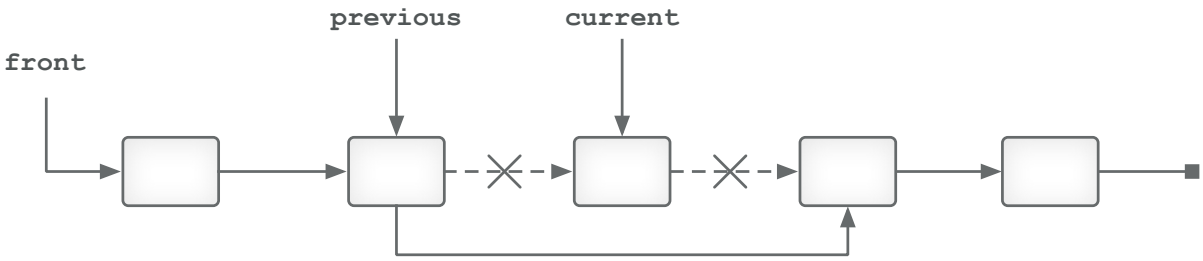

```
1. insertAfter(node, current)
2.     node.next = current.next
3.     current.next = node
```

Remover um nó na lista ligada

Para removermos um nó numa lista acabamos por ter as mesmas duas situações que tínhamos para o caso anterior: remover um nó na cabeça (`front`) ou no meio da lista ligada.

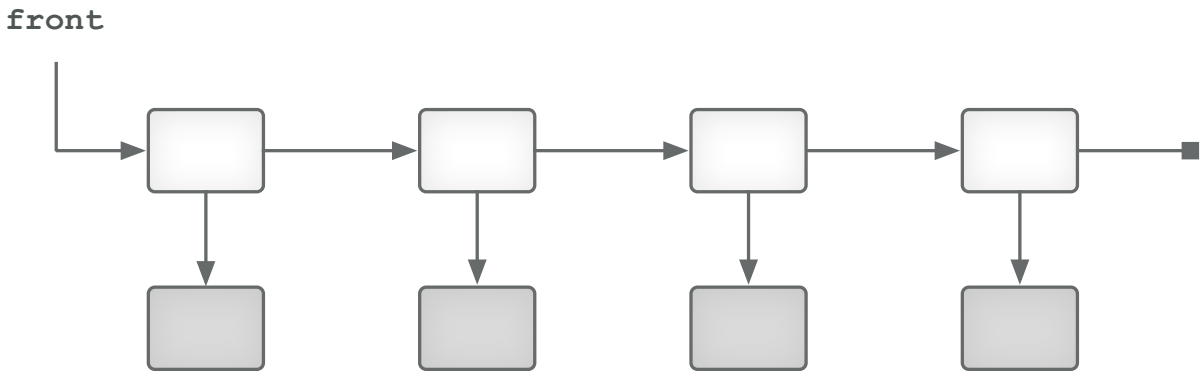



```
1. removeFirst()
2.     front = node.next
```




```
1. removeAfter(previous, current)
2.     previous.next = current.next
```

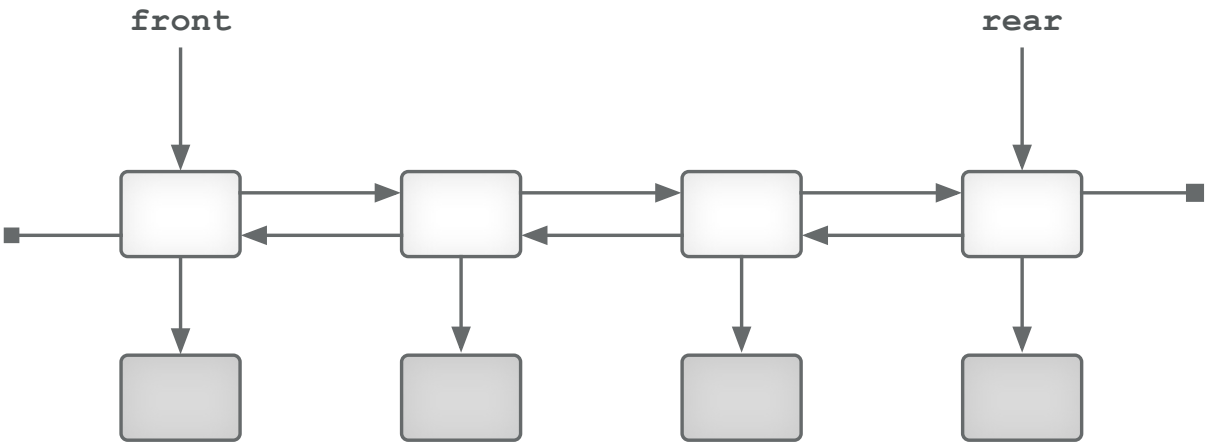
Os diagramas apresentados até agora não tratam bem a realidade, efetivamente como estamos a trabalhar segundo o paradigma orientado a objetos o elemento é também um objeto portanto terá de existir um referência do nó para o objeto elemento como apresentado na figura seguinte.



Obviamente os algoritmos apresentados não sofrem qualquer alteração já que se trata apenas do objeto que mantém o elemento a ser armazenado. Os algoritmos apresentados não manipulam esse objeto.

5.2. Listas Duplamente Ligadas

As listas duplamente ligadas são muito semelhantes às listas simplesmente ligadas. A única diferença é que cada nó tem uma referência tanto para o próximo elemento quanto para o anterior na lista.

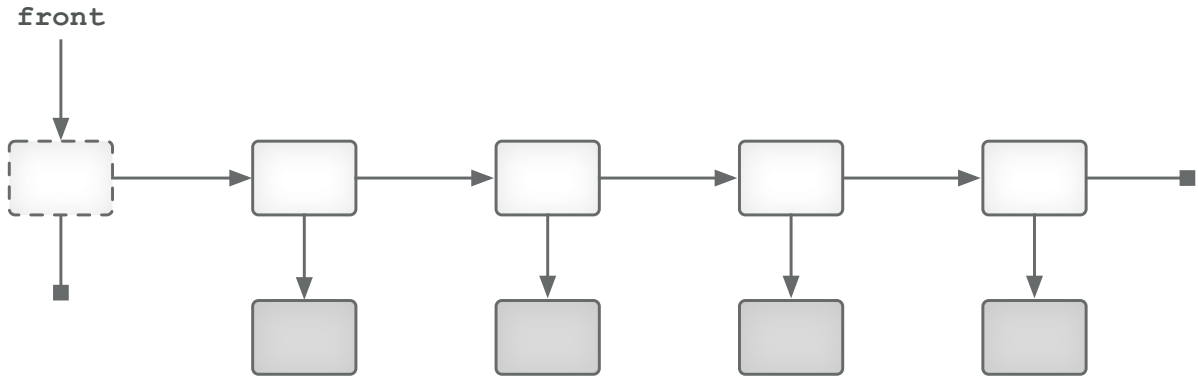


Uma lista deste tipo permite a pesquisa da lista em ambas as direções (para retroceder numa lista simplesmente ligada é necessário voltar ao início e percorrer novamente desde o início). Neste caso, a estrutura do nó como passa a ter dois *links* inverter o sentido é imediato.

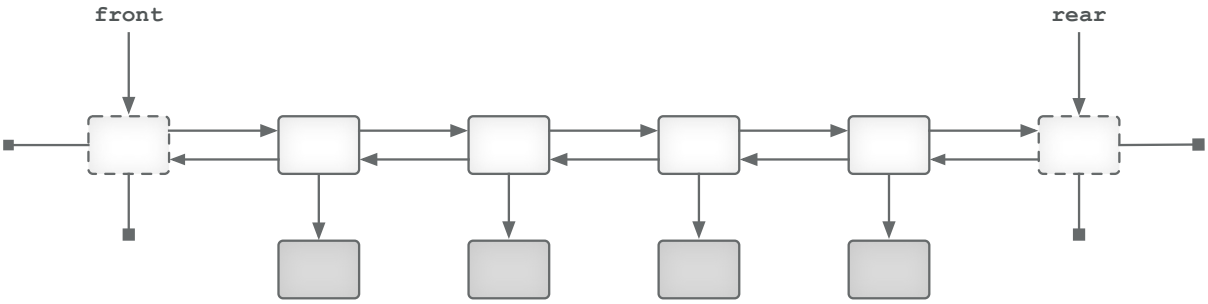
5.3. Nós Sentinela

Existem variações sobre a implementação de listas ligadas que podem ser úteis em situações particulares. Uma dessas soluções é o uso do nós sentinelas ou nós fictícios em cada extremidade da lista. Esta prática elimina os casos especiais de inserir ou eliminar o primeiro ou último nó.

No caso de uma lista simplesmente ligada apenas temos um nó sentina, que representa o nó `front` como podemos ver na figura seguinte.



Já para o caso de uma lista duplamente ligada temos dois nós sentinela, um que representa o nó `front` e outro que representa o nó `rear` como podemos ver na figura seguinte.



Como se consegue depreender dos diagramas apresentados a complexidade dos algoritmos passa a ser muito menor já que deixa de haver os casos particulares de "inicio" e "fim" da lista. Qualquer inserção ou remoção passa a ser de um nó interno.

5.4. Utilização de Listas Ligadas

As listas ligadas são boas para usar quando temos um número desconhecido de itens para armazenar. Usar uma estrutura de dados como um *array* exigiria que fosse especificado o tamanho antecipadamente. Exceder esse tamanho envolve invocar um algoritmo de redimensionamento que tem um tempo de execução linear. Também devemos usar listas ligadas quando pretendemos apenas remover nós no início ou no final da lista de forma a ser mantido um tempo de execução constante. Para isso temos que manter uma referência para esses nós (início e fim da lista), mas a sobrecarga em termos de memória irá compensar se esta operação for executada muitas vezes.

As listas ligadas não são adequadas para a inserção aleatória, aceder a nós através de um índice assim como para pesquisas. As listas simplesmente ligadas devem ser usadas quando pretendemos realizar apenas inserções básicas. Em geral, listas duplamente ligadas são mais adequadas para operações não triviais numa lista ligada. O uso de uma lista duplamente ligada é recomendado quando necessitamos percorrer um conjunto de elementos para frente e para trás. Na maioria dos casos este requisito está presente.

5.5. Exercícios Propostos

Exercício 1

Este primeiro exercício consiste na criação de uma lista ligada (*LinkedList*). A lista ligada deverá possuir as seguintes operações: `Add` e `Remove`.

Para que seja possível verificar a integridade da lista ligada crie uma função que imprima todos os elementos da lista.

Atenção: Não confundir com a classe `java.util.LinkedList` (disponível na plataforma de coleções do Java)

Exercício 2

Criar uma implementação de lista ligada que use nós sentinela. Esta implementação deverá também possuir as operações `Add` e `Remove`.

Para que seja possível verificar a integridade da lista ligada crie uma função que imprima todos os elementos da lista.

Exercício 3

Consegue perceber a diferença entre as duas implementações anteriores? Observe bem as operações implementadas.

Exercício 4

Criar uma lista duplamente ligada (`DoublyLinkedList`) capaz de realizar as seguintes operações:

- Inserir um nó na cabeça;
- Remover o primeiro nó de uma lista;
- Remover o último nó de uma lista;
- Indicar se a lista está vazia ou não;
- Percorrer e imprimir todos os elementos da lista.

Exercício 5

Responda às seguintes questões:

- Quais são os componentes básicos que compõem uma lista ligada?

- Para que serve um nó numa lista ligada?
- Qual a diferença entre um *array* e uma lista ligada?
- Qual a diferença entre uma lista ligada e uma lista duplamente ligada?

Exercício 6

Criar as seguintes operações na (`DoublyLinkedList`):

- Devolver um *array* com todos os elementos;
- Devolver um *array* com todos os elementos até uma data posição;
- Devolver um *array* com todos os elementos depois de uma dada posição;
- Devolver um *array* com todos os elementos entre um intervalo de posições.

Exercício 7

Devolver uma nova `DoublyLinkedList` com apenas os elementos pares (Sempre que for instanciada com tipos inteiros).

Exercício 8

Alterar a `DoublyLinkedList` de forma que esta seja capaz de permitir saber quantos elementos iguais a um dado elemento que é passado por parâmetro estão presentes na `DoublyLinkedList`. Remover todos esses elementos e manter a integridade da `DoublyLinkedList`.