Estruturas de Dados Fundamentais

Arranjos e listas encadeadas

Prof. Marcelo de Souza

45RPE – Resolução de Problemas com Estruturas de Dados
Universidade do Estado de Santa Catarina

Arranjos

Ou seja, *arrays*/vetores

Arranjos são **estruturas de dados sequenciais**, armazenando sequências finitas e ordenadas de valores de um mesmo tipo. Por exemplo:

- **Números:** 1, 2, 4, 5, 7, 8, ...
- **Strings:** "Brasil", "Alemanha", "Croácia", ...
- **Veículos:** ("Corcel", 1977), ("Fusca", 1968), ("Passat", 1984), ...

Arranjos

Ou seja, *arrays*/vetores

Arranjos são **estruturas de dados sequenciais**, armazenando sequências finitas e ordenadas de valores de um mesmo tipo. Por exemplo:

- **Números:** 1, 2, 4, 5, 7, 8, ...
- **Strings:** "Brasil", "Alemanha", "Croácia", ...
- **Veículos:** ("Corcel", 1977), ("Fusca", 1968), ("Passat", 1984), ...

A principal característica dos arranjos é a **alocação contígua** em memória.

- **Vantagens:**
  - Fácil de usar;
  - Acesso rápido (tempo constante).
- **Desvantagens:**
  - Tamanho fixo (aumentar implica copiar elementos);
  - Inserção e remoção [interna] custosas (*shift* de elementos).

Listas simplesmente encadeadas

Encadeamento

Queremos uma estrutura de dados dinâmica que permita **expandir** e **contrair** com eficiência.

Listas simplesmente encadeadas

Encadeamento

Queremos uma estrutura de dados dinâmica que permita **expandir** e **contrair** com eficiência.

Lista encadeada: coleção de nodos formados em uma sequência linear.

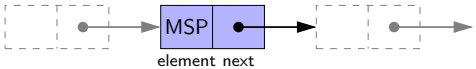
Listas simplesmente encadeadas

Encadeamento

Queremos uma estrutura de dados dinâmica que permita **expandir** e **contrair** com eficiência.

Lista encadeada: coleção de nodos formados em uma sequência linear.

Lista simplesmente encadeada: cada nodo armazena os dados do elemento e uma referência ao próximo nodo. A alocação em memória **não é contígua**.



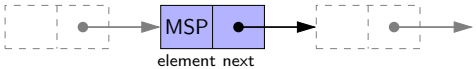
Listas simplesmente encadeadas

Encadeamento

Queremos uma estrutura de dados dinâmica que permita **expandir** e **contrair** com eficiência.

Lista encadeada: coleção de nodos formados em uma sequência linear.

Lista simplesmente encadeada: cada nodo armazena os dados do elemento e uma referência ao próximo nodo. A alocação em memória **não é contígua**.



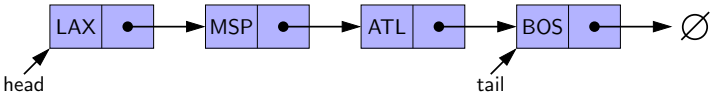
Benefícios:

- Tamanho dinâmico;
- Consumo de memória dinâmico;
- Fácil inserção e remoção de elementos.

Listas simplesmente encadeadas

Encadeamento

Exemplo: uma lista simplesmente encadeada para armazenar aeroportos dos EUA.



Elementos:

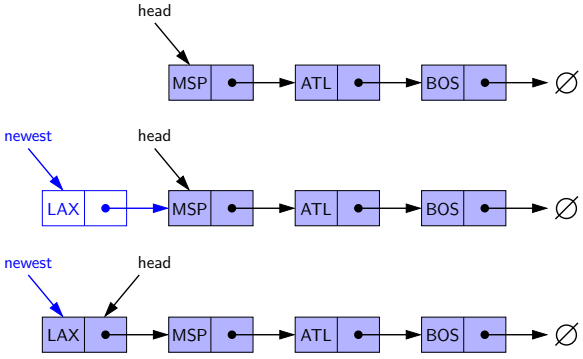
- `head`: referência ao primeiro elemento da lista;
- `tail`: referência ao último elemento da lista;
- O próximo elemento do último elemento aponta para `null`.

Listas simplesmente encadeadas

Operações

Inserção no início

```
newest.next ← head
head ← newest
```



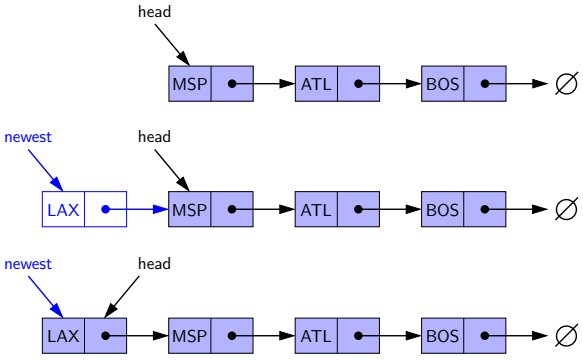
Listas simplesmente encadeadas

Operações

Inserção no início

```
newest.next ← head
head ← newest
```

Operação em $\mathcal{O}(1)$.

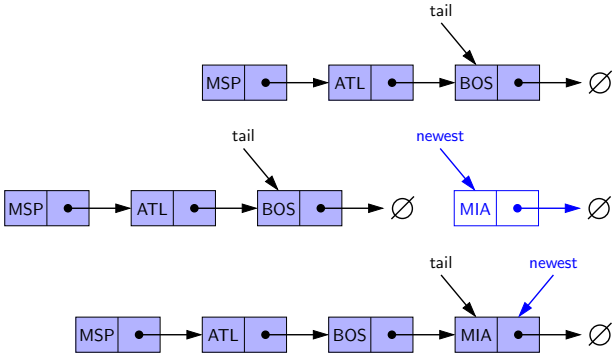


Listas simplesmente encadeadas

Operações

Inserção no final

```
tail.next ← newest
tail ← newest
```



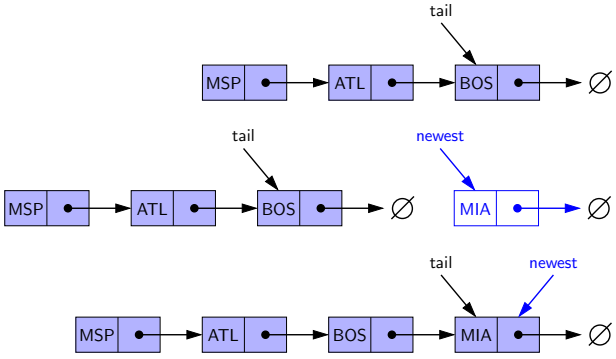
Listas simplesmente encadeadas

Operações

Inserção no final

```
tail.next ← newest
tail ← newest
```

Operação em $\mathcal{O}(1)$.

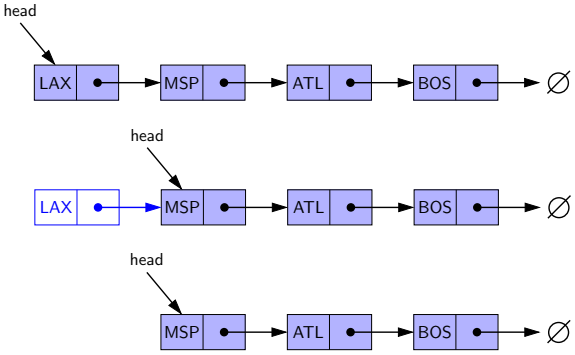


Listas simplesmente encadeadas

Operações

Remoção do início

```
head ← head.next
```



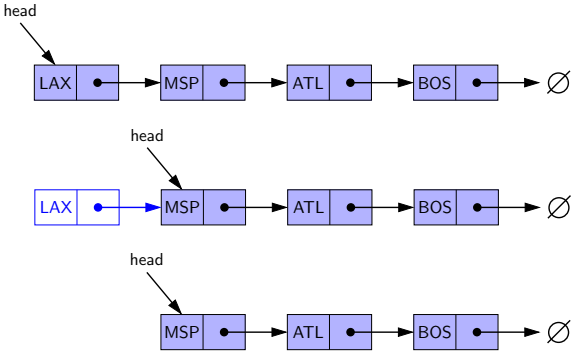
Listas simplesmente encadeadas

Operações

Remoção do início

`head` ← `head.next`

Operação em $\mathcal{O}(1)$.



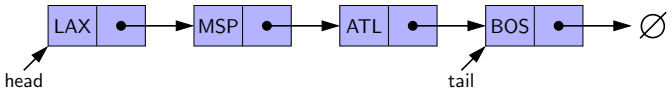
Listas simplesmente encadeadas

Operações

E se quisermos **inserir** ou **remover** em/de uma **posição arbitrária**?

1. Percorre a lista, até chegar no nodo da posição desejada.
2. Faz a inserção ou remoção, atualizando as referências.

Operação em $O(n)$, por conta da necessidade de percurso.



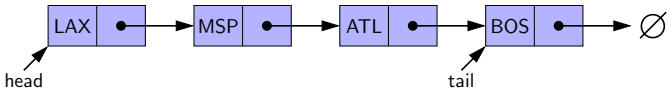
Listas simplesmente encadeadas

Operações

E se quisermos **inserir** ou **remover** em/de uma **posição arbitrária**?

1. Percorre a lista, até chegar no nodo da posição desejada.
2. Faz a inserção ou remoção, atualizando as referências.

Operação em $O(n)$, por conta da necessidade de percurso.



E se quisermos **remover** do **final** da lista?

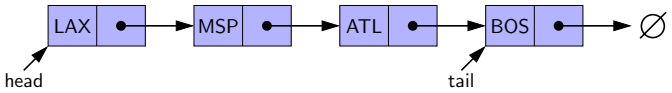
Listas simplesmente encadeadas

Operações

E se quisermos **inserir** ou **remover** em/de uma **posição arbitrária**?

1. Percorre a lista, até chegar no nodo da posição desejada.
2. Faz a inserção ou remoção, atualizando as referências.

Operação em $O(n)$, por conta da necessidade de percurso.



E se quisermos **remover** do **final** da lista?

1. Percorre a lista até o penúltimo nodo.
2. Atualiza sua referência `next` para `null` e a referência do `tail`.

Operação em $O(n)$, por conta da necessidade de percurso.

Listas duplamente encadeadas

Conceito e motivação

Problemas do encadeamento simples:

- Não conseguimos **remover o último nodo** de forma eficiente.
  - Precisamos atualizar a referência `next` do nodo anterior.
  - Para chegar no penúltimo, precisamos **percorrer a lista**.
- Remover um nodo (que não seja o primeiro) tendo apenas a sua referência é custoso.

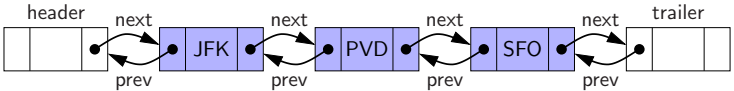
Listas duplamente encadeadas

Conceito e motivação

Problemas do encadeamento simples:

- Não conseguimos **remover o último nodo** de forma eficiente.
  - Precisamos atualizar a referência `next` do nodo anterior.
  - Para chegar no penúltimo, precisamos **percorrer a lista**.
- Remover um nodo (que não seja o primeiro) tendo apenas a sua referência é custoso.

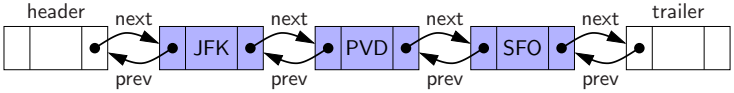
Lista duplamente encadeada: cada nodo mantém a referência do anterior (`prev`) e do próximo (`next`) nodos.



Listas duplamente encadeadas

Sentinelas

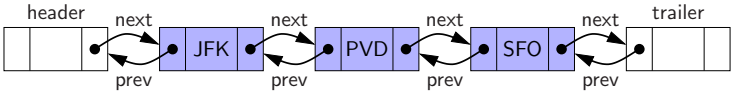
Na implementação dessas listas, usamos uma técnica muito útil: uso de **nodos sentinelas**. Trata-se de nodos vazios ("fictícios") no início (`header`) e no fim (`trailer`) da lista.



Listas duplamente encadeadas

Sentinelas

Na implementação dessas listas, usamos uma técnica muito útil: uso de **nodos sentinelas**. Trata-se de nodos vazios ("fictícios") no início (`header`) e no fim (`trailer`) da lista.



Facilidades (na implementação):

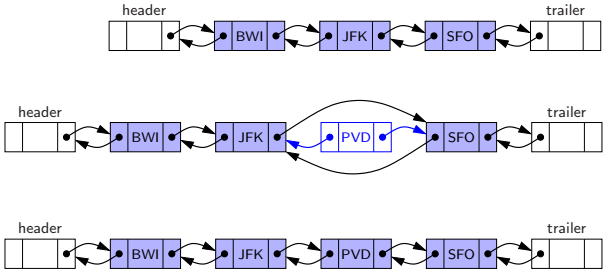
- Certeza de que cada nodo possui dois vizinhos;
- Toda inserção será entre dois nodos.
  - Nunca inserimos no verdadeiro início ou fim;
  - Casos excepcionais (lista vazia ou com um nodo) não acontecem.

Listas duplamente encadeadas

Operações

Inserção arbitrária (entre `A` ↔ `B`)

```
newest.prev ← A
newest.next ← B
A.next ← newest
B.prev ← newest
```

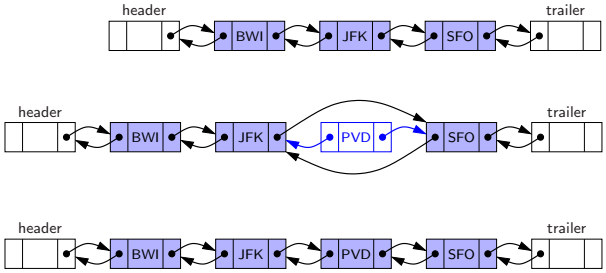


Listas duplamente encadeadas

Operações

Inserção arbitrária (entre `A` ↔ `B`)

```
newest.prev ← A
newest.next ← B
A.next ← newest
B.prev ← newest
```



A inserção tendo as **referências** dos nodos vizinhos exige $\mathcal{O}(1)$.

A inserção nas **extremidades** exige $\mathcal{O}(1)$.

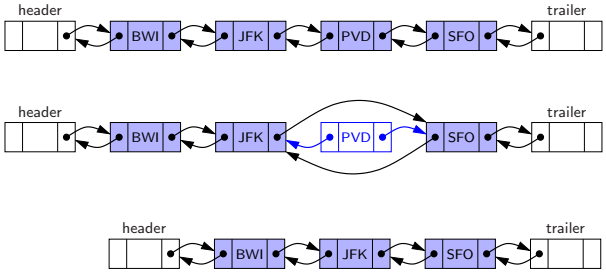
A inserção tendo somente a **posição** desejada exige $\mathcal{O}(n)$, dada a necessidade de percurso.

Listas duplamente encadeadas

Operações

Remoção arbitrária (do nodo `X`)

```
X.prev.next ← X.next
X.next.prev ← X.prev
```

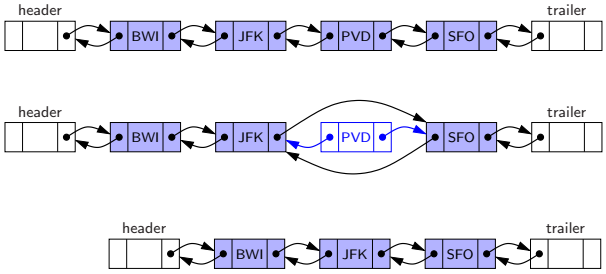


Listas duplamente encadeadas

Operações

Remoção arbitrária (do nodo `X`)

```
X.prev.next ← X.next
X.next.prev ← X.prev
```



A remoção tendo a **referência** do nodo a ser removido exige $\mathcal{O}(1)$.

A remoção das **extremidades** exige $\mathcal{O}(1)$.

A remoção tendo somente a **posição** desejada exige $\mathcal{O}(n)$, dada a necessidade de percurso.

45RPE – Resolução de Problemas com Estruturas de Dados
Prof. Marcelo de Souza