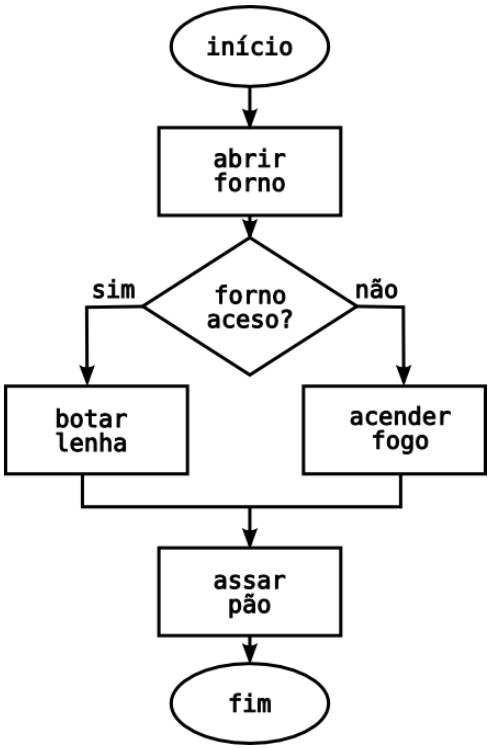
O que é um Algoritmo?

Um algoritmo é uma sequência finita de passos bem definidos usada para resolver um problema. Eles estão presentes em todas as áreas da computação, desde funções simples até sistemas complexos. Quando vários algoritmos trabalham juntos, formam um **caso de uso**, oferecendo uma funcionalidade ao usuário. No desenvolvimento de software, entender e avaliar a complexidade dos algoritmos é essencial para garantir performance e eficiência.



Complexidade de Algoritmos

A complexidade algorítmica se refere aos recursos necessários para executar um algoritmo, como tempo de processamento e uso de memória. Um exemplo histórico interessante é o da NASA, onde, antes dos computadores IBM 700/7000, os cálculos eram feitos manualmente por "computadores humanos". Atualmente, em um cenário de Big Data e computação em nuvem, os algoritmos precisam ser eficientes e escaláveis para lidar com grandes volumes de dados.

Notações Assintóticas

A eficiência de um algoritmo é medida com **notações assintóticas**, que descrevem o comportamento do algoritmo conforme o tamanho da entrada cresce. As principais notações são:

- **Big-O (O)**: Limite superior, o pior caso.
- **Big-Ω (Ômega)**: Limite inferior, o melhor caso.
- **Big-Θ (Theta)**: Caso médio, onde o melhor e pior caso são iguais.

Exemplos de Complexidade

- **O(1)** – Constante: O tempo de execução não depende do tamanho da entrada.
- **O(n)** – Linear: O tempo cresce proporcionalmente ao tamanho da entrada.
- **O(n^2)** – Quadrática: O tempo cresce exponencialmente com o aumento dos dados.

Contagem de Operações

A contagem de operações é feita com base em operações primitivas, como comparações e atribuições. Para algoritmos mais simples, a análise é direta, enquanto algoritmos mais complexos, como os recursivos, podem ser analisados por meio de equações de recorrência.

Exemplos Práticos

- **Cálculo da média de uma lista**: Complexidade O(n), já que depende do número de elementos.
- **Bubble Sort**: Complexidade O(n^2) devido ao duplo laço for.

Benchmarking

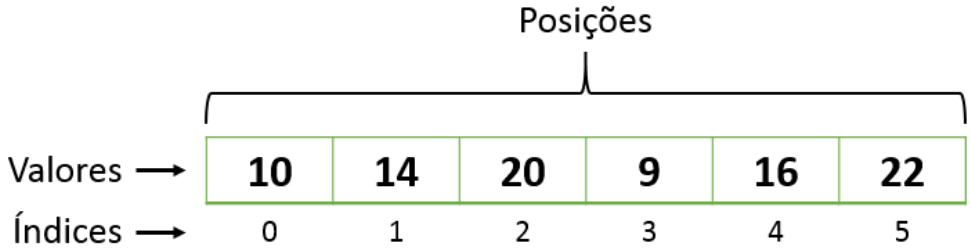
O benchmarking é a prática de testar algoritmos com diferentes entradas para entender seu desempenho em diferentes cenários. Isso é essencial para decidir qual algoritmo usar em cada situação.

Vetores, Buscas e Ordenação

Vetores-(Arrays)

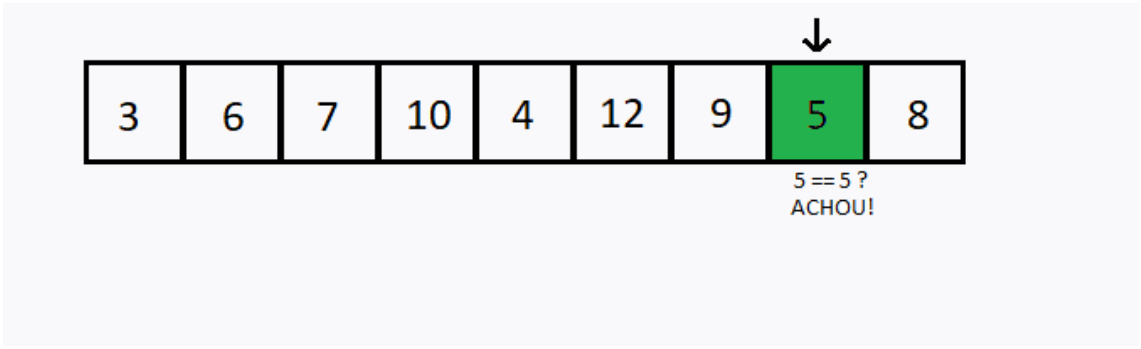
Vetores são estruturas unidimensionais, onde os elementos são armazenados em posições numeradas, geralmente começando pelo índice 0. Acessos aos dados são feitos de maneira eficiente usando `vet[i]`. Quando precisamos preencher ou embaralhar vetores,

usamos algoritmos como o de **Fisher-Yates** para gerar números aleatórios ou reorganizar os dados de forma aleatória.



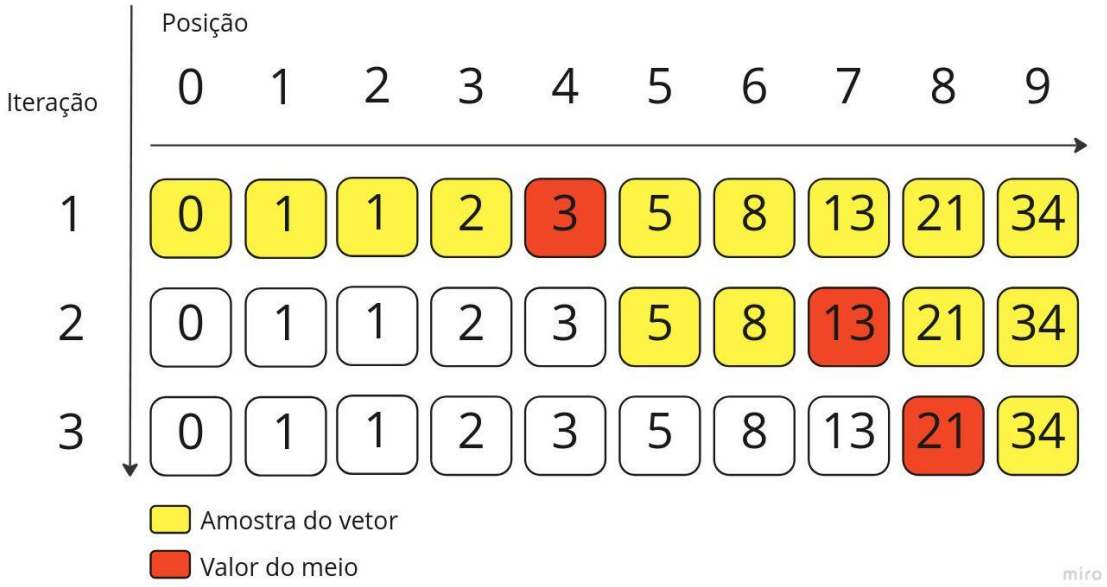
Busca-Sequencial-(Linear)

A busca sequencial percorre o vetor elemento por elemento até encontrar o valor desejado. É simples, mas tem complexidade O(n) no pior e no caso médio, já que pode ser necessário verificar todos os elementos.



Busca-Binária

A busca binária é mais eficiente, mas exige que o vetor esteja ordenado. Ela compara o valor buscado com o elemento central do vetor e, dependendo do resultado, busca à esquerda ou à direita, reduzindo significativamente o número de comparações.



Deslocamento em Vetores

Para inserir ou remover elementos de um vetor, precisamos deslocar outros elementos, o que pode ser feito com laços for. Isso é importante para manter a integridade da estrutura de dados, especialmente quando os elementos são inseridos ou removidos de posições específicas.

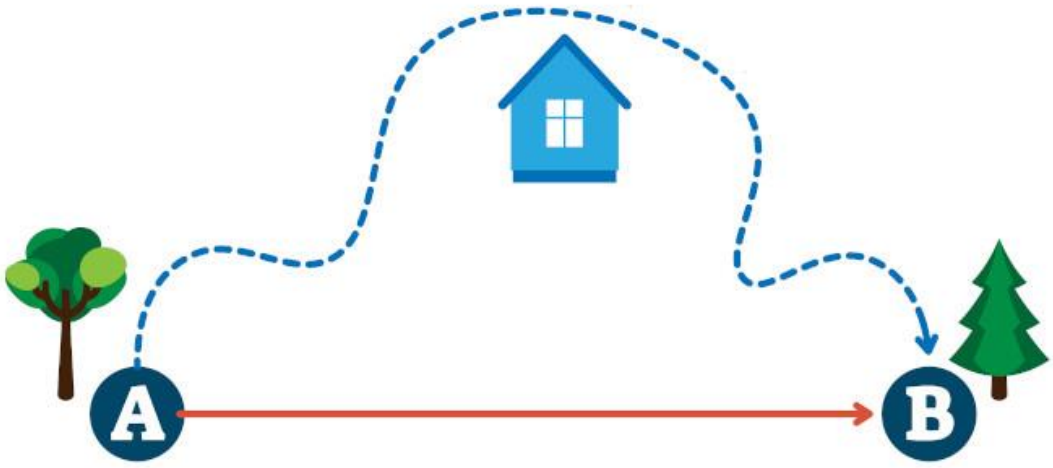


Tabela Hash

O que é e por que usar?

Uma tabela hash é uma estrutura de dados que oferece busca rápida através de chaves. A chave funciona como um "endereço" único, permitindo o acesso direto aos dados. Isso é essencial quando há grandes volumes de dados e muitas buscas frequentes.

Problema da Memória

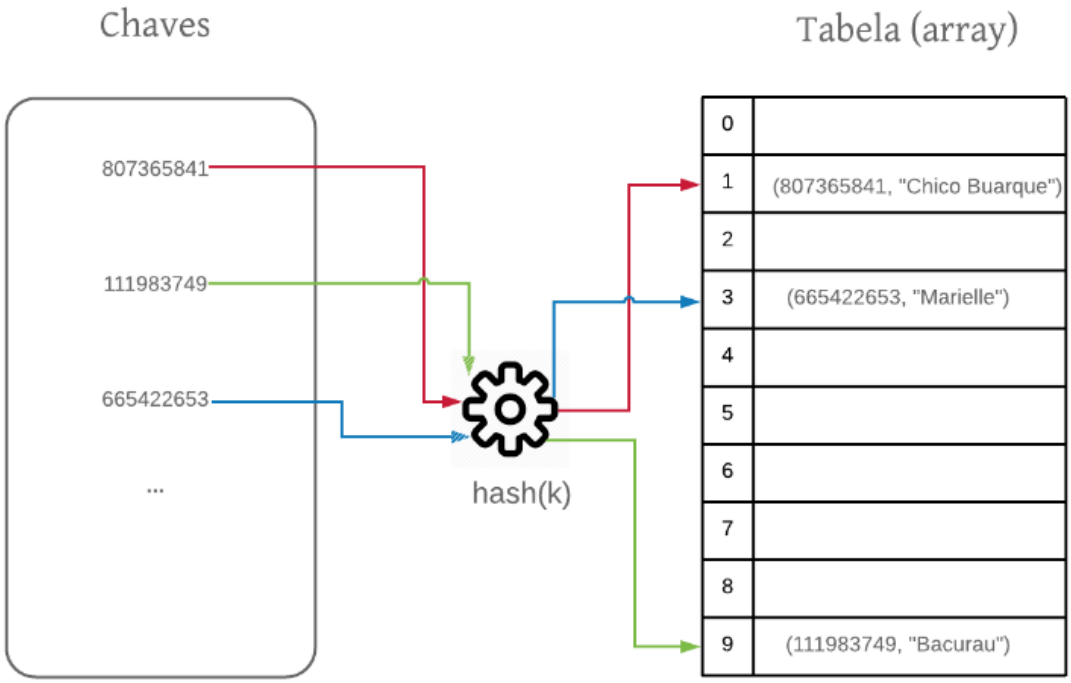
Se usarmos chaves grandes, podemos precisar de tabelas muito grandes para armazenar dados, o que se torna ineficiente. A solução para isso é a **função hash**, que mapeia as chaves para posições menores e mais eficientes, calculando o resto de uma divisão.

Colisões

Uma colisão ocorre quando duas chaves diferentes resultam no mesmo índice. Para resolver isso, podemos usar sondagem linear (procurando a próxima posição disponível) ou listas encadeadas (armazenando múltiplos itens na mesma posição).

Fator de Carga

O fator de carga mede a ocupação da tabela e é calculado dividindo o número de elementos pelo tamanho da tabela. Para garantir boa performance, é importante manter o fator de carga controlado e usar um tamanho primo para a tabela.



Filas Estáticas Sequenciais

Conceito de fila

Uma fila é uma estrutura de dados do tipo **FIFO** (First In, First Out), ou seja, o primeiro elemento a entrar é o primeiro a sair. As operações principais são **enqueue(x)** para adicionar um item e **dequeue()** para remover o primeiro item.

Fila Estática vs Circular

A fila **estática simples** tem o problema dos "espaços mortos" quando elementos são removidos, tornando o uso da memória ineficiente. Já a **fila circular** resolve esse problema ao reutilizar as posições livres quando a fila chega ao final, melhorando o aproveitamento da memória.

Implementação em Java

Na implementação de uma fila estática, usamos um vetor para armazenar os dados, com dois índices principais: **head** e **tail**, que controlam o início e o fim da fila. As verificações de **fila cheia** e **fila vazia** são essenciais para o controle da estrutura.