



Estrutura de Dados ✓

Material das Aulas

Material Complementar

Introdução

Estrutura de Dados Clássicas

Estrutura de Dados e Algoritmo para classificar e organizar a recuperação da informação

Pilha I

O que são Arrays/Vetores?

O que são Pilhas

LIFO

Exemplificando uma pilha

Push

Obs.:

Pilhas II

Implementação da Pilha no Javascript

Filas

O que são filas?

Exemplificando uma fila

FIFO

Listas encadeadas

O que são?

Características

Busca I - Sequencial

Busca sequencial

Implementação com Javascript

Busca II - Binária

Complexidade é de ordem logarítmica \log de O

Implementação com Javascript

Algoritmos de Ordenação I

Bubble Sort

Implementação com Javascript

Insertion Sort

Selection Sort

Quick Sort

Algoritmos de Ordenação II

Tabela Hash

Árvore Balanceada

Implementação em Javascript

Material das Aulas

- [Slide](#)

Material Complementar

Estrutura de Dados (A famosa ED que todo dev tem que aprender) // Dicionário do Programador


Muito se fala, nós inclusive, que antes de sair aprendendo uma linguagem é necessário conhecer mais sobre lógica, algoritmos e Estrutura de Dados (ED). Sim e...

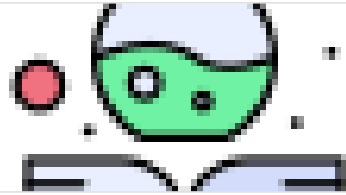
📺 https://www.youtube.com/watch?v=EIF1M7myAyY&ab_channel=C%C3%B3digoFonteTV



Estrutura de dados, para que servem e por que tão temidas?

As estruturas de dados nada mais são do que formas para armazenar dados na memória, as formas de armazenamento podem variar, podendo ser tanto de maneira sequencial ou não, como é o caso de uma estrutura falada mais a frente.

 <https://gasparbarancelli.com/post/estrutura-de-dados-para-que-servem-e-por-que-tao-temidas>



Introdução

Estrutura de Dados Clássicas

- Se preocupa com a maneira que as informações são ordenadas
- Listas, filas e pilhas

Estrutura de Dados e Algoritmo para classificar e organizar a recuperação da informação

- Buscar conjuntos de valores de forma rápida numa base de dados grande.
- Qual forma de fazer isso rapidamente, qual algoritmo eu devo usar?
- Existem técnicas e algoritmos para fazermos isso de forma otimizada.
- Temos busca sequencial, busca binária.
- Temos algoritmos de classificação e ordenação envolvendo estrutura de dados: algoritmos bubble sort etc, e estruturas como hash e árvore.

Pilha I

- Quando falamos de estrutura de dados em LP falamos de vetores ou arrays.

O que são Arrays/Vetores?

- É um variável que permite a inserção de vários valores por vez

O que são Pilhas

- Temos coleções de valores e regras de como inserir esses valores dentro dessa estrutura.
- Ela é um tipo de dado abstrato.
- Pilhas são estruturas de dados que armazenam os elementos em um formato sequencial, **empilhando** um item acima do outro (imagine uma pilha de livros, por exemplo).
- Estas estruturas permitem “empilhar” os itens que serão armazenados e “desempilhar” estes elementos da pilha quando precisarmos removê-lo. Sempre que um novo elemento é inserido (ou empilhado) damos a ele o nome de “topo”, pois é o primeiro elemento ao qual teremos acesso.

LIFO

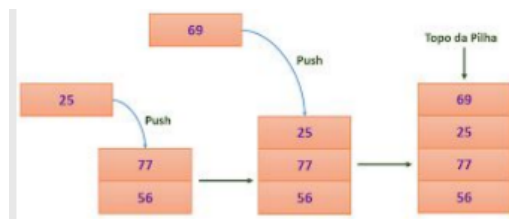
- Segue um padrão conhecido como **LIFO (Last In First Out)**, onde o último a entrar será o primeiro a sair.
Imagine uma pilha de livros, sempre que um livro é “empilhado” sob o outro, este último livro empilhado é o mais próximo (ou o topo da pilha) e, caso precisarmos remover um livro, é o livro do topo que será removido da estrutura.

Exemplificando uma pilha

- Vamos imaginar um cenário onde diversos blocos precisam ser alocados e armazenado em uma pilha.
- Exemplos práticos: empilhamento de materiais.
- O primeiro elemento “Bloco 56” foi adicionado à nossa pilha, representando seu primeiro elemento (Primeiro nó).

Push

- À esta funcionalidade de inserção de elementos damos o nome de push.
- Logo, estaremos inserindo em seguida o próximo elemento “Bloco 56”. Este elemento será então inserido logo acima do nosso elemento anterior (“Bloco 56”), conforme imagem ilustrando:
- A vazão destes elementos vai tratar sempre da seguinte maneira, o primeiro item a sair, no caso é sempre o último que foi empilhado.



Obs.:

- Push insere um elemento, e pop retira um elemento da pilha.
- Temos um vetor para armazenar o **valor**, e um **índice (index)** que controla a posição

Pilhas II

Implementação da Pilha no Javascript

- Toda coleção começa com o índice 0, que é o primeiro elemento.
- Ordem de empilhar é a ordem inversa de desempilhar.

Função Push

- Adicionar um elemento no topo da Pilha

```
function push(num){
  if (topo < MAX){
    topo = topo + 1;
    elementos[topo] = num;
  }
  else{
    console.log("Pilha esta cheia");
  }
}
```

Função Pop

- Retira um elemento do topo da pilha

```
function pop(){
  if (topo !== -1){
    let num = elementos[topo];
    topo = topo - 1;
    return num;
  }
  else{
    console.log("Pilha esta vazia!");
  }
}
```

Filas

O que são filas?

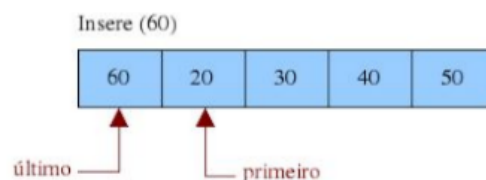
- Uma estrutura de dados é uma maneira de armazenar e relacionar conjuntos de informações de forma organizada e, na maioria das vezes, sequencial.
- Estas estruturas são muito importantes quando precisamos armazenar um conjunto de dados para ser utilizado em um determinado software.
- Na computação, há diversos tipos de estruturas de dados que podem ser utilizadas para diferentes fins. Vimos nos artigos anteriores o que é e como funciona a Estrutura de Dados Lista e o que é e como funciona a Estrutura de dados Pilha.
- Neste capítulo veremos o que é e como funciona a Estrutura de Dados Fila.

Exemplificando uma fila

- Fila são estruturas de dados bastante utilizadas na computação, onde o primeiro elemento a ser inserido, será também o primeiro a ser retirado.
- Desta forma, serão adicionados elementos no fim e removê-los pelo início.

FIFO

- A estrutura de dados fila seguem um padrão conhecido como FIFO (first-in first-out), onde o primeiro a entrar é o primeiro a sair.
- Imagine o exemplo de um fila do banco, onde diversos usuários a compõe em uma manhã de segunda feira.



Listas encadeadas



Com pilha e fila trabalhamos no modelo de vetor e com tamanhos limitados (capacidade máxima).



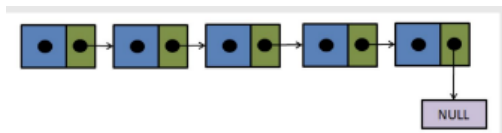
Implementamos pilhas e filas com **listas**.

O que são?

- Uma lista encadeada é uma representação de uma sequência de objetos, todos do mesmo tipo, na memória RAM (= random access memory) do computador.
- Cada elemento da sequência é armazenado em uma célula da lista: o primeiro elemento na primeira célula, o segundo na segunda, e assim por diante.
- Temos várias formas de ordenação e organização
- Pilhas, filas e listas tratam sobre conjuntos e conjuntos tem critérios para colocarmos elementos da forma que queremos.

Características

- Quando lidamos com arrays, devemos saber de antemão ou em tempo de compilação qual o tamanho ou quantidade de bytes a ser alocados para o array.
- Para estes casos pode-se alocar a memória necessária via alocação dinâmica de memória. No entanto ainda existem outros casos em que mais memória deve ser alocada para armazenar os dados do programa a medida que o programa vai sendo utilizado.
- Este é um dos casos em que a utilização de listas ligadas é mais indicada do que a utilização de arrays estáticos ou dinâmicos.
- Outra vantagem das listas ligadas é que não existe nenhuma restrição indicando que as células que as compõem devem estar alocadas sequencialmente na memória tal como ocorre com arrays ou memória alocada dinamicamente via malloc.



Busca I - Sequencial

- Busca de elementos em conjuntos.
- Qual é a melhor forma de buscar elementos?

Busca sequencial

- A partir de um determinado conjunto de valores queremos encontrar qual posição está aquele valor.
- Caso não encontremos, retornamos o índice -1, indicando que não o elemento não está na lista

Implementação com Javascript

- Fizemos um `for` para buscar em cada índice, se existe o valor que queremos.
- Caso não haja retorna o `else`

```

var valores = [5, 8, 10, 22, 45, 38];

function busca(num) {
  for (i = 0; i < 6; i++) {
    if (num == valores[i]) {
      return i;
    }
  }
  return -1;
}

// usando a nossa ferramenta de busca

console.log(busca(10));
console.log(busca(50));

```

Busca II - Binária

- Para conjuntos pequenos nossa Busca I resolve, mas para registros mais extensos ela acaba ficando não performática.
- Busca binária é uma busca que é muito mais eficiente, mas exige que os valores devam estar **ordenados**.
- Buscamos sempre o elemento do meio, se for retorno o cara
- Se for maior busco do meio para cima, se menor do meio para baixo.
- Vou reduzindo sempre a metade de acordo com o retorno, até achar o elemento que queremos, ou não reduzimos nada se não acharmos.
- Vamos reduzindo a faixa (espectro) para encontrar o algoritmo

Complexidade é de ordem logarítmica log de O

- No pior caso, o elemento não é encontrado em 8 passos num conjunto de 64.
- Fazemos tudo na base 2 e conseguimos encontrar quantos passos retornamos ou não o elemento
- O único requisito é que esteja ordenado, para **busca binária**.

Implementação com Javascript

```

function buscaBin(num) {
  let inicio, fim;
  let meio;
  inicio = 0;
  fim = 9;
  while (inicio <= fim) {
    meio = parseInt((inicio + fim) / 2);
    if (num == valores[meio]) {
      return meio;
    } else {
      if (num > valores[meio]) {
        inicio = meio + 1;
      } else {
        fim = meio - 1;
      }
    }
  }
  return -1;
}

```

Algoritmos de Ordenação I

- Existem vários algoritmos de ordenação de valores

Bubble Sort

- Dado um conjunto de valores, pegamos o primeiro conjunto de valor em relação ao seu próximo
- Se o valor que estamos comparando é maior que o próximo, então trocamos o valor
- Assim, vamos ordenando do leve ao pesado (do menor para o maior).

Implementação com Javascript

- Comparamos um elemento sempre com o seu próximo, até chegar o penúltimo valor.
- Esse **vetor ordenado**, podemos aplicar a **busca binária** nele.

```
for (vezes = 0; vezes < 8; vezes++) {  
  for (pos = inicio; pos < fim - 1 - vezes; pos++) {  
    if (valores[pos] > valores[pos + 1]) {  
      tmp = valores[pos];  
      valores[pos] = valores[pos + 1];  
      valores[pos + 1] = tmp;  
    }  
  }  
}
```

Insertion Sort

- Pega uma posição aleatória e desloca até o ponto que queremos

Selection Sort

- Percorre o vetor, acha o maior e coloca na última posição
- Vai achando os maiores e colocando maior para o menor.

Quick Sort

- Quebra os vetores em sub-vetores de modo que eles estejam desordenados, porém a metade é sempre maior que o nosso elemento central, e a outra metade sempre menor que nosso elemento central.
- Lembra a busca binária de elementos.



Primeiro precisamos ordenar o vetor, para depois fazer a busca de um elemento

Algoritmos de Ordenação II

- Podemos ter uma estrutura de dados que já insere os vetores e fazemos a busca
- Ou seja, ela já ordena e permite pesquisarmos

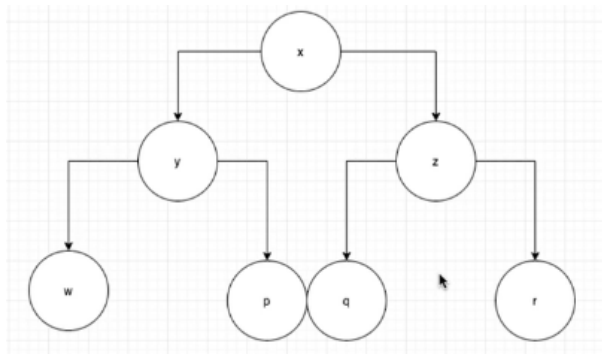
Tabela Hash

- Vetores de palavra-chave e valores, para que se tivermos boas palavras-chaves a tendência é que ela não se repita.
- Temos um valor a ser armazenado e transformamos em uma chave (descreve unicamente aquele registro) e a chave é convertida para uma posição através de um algoritmo.
- Na hora de buscar sabemos exatamente qual é o vetor através da função de conversão.

- Mapeamento da informação para posição.
- Acessamos o elemento num único passo.
- É muito performático a busca e armazenamento.

Árvore Balanceada

- Ela é muito organizado
- O elemento está sempre localizado no Nó Raiz.
- Se for o elemento que buscamos e estiver já no Nó Raiz, já devolvemos
- Se não for, e for menor, esse elemento sempre estará no lado **esquerdo**.
- Se não, e for maior, ele vai estar na descendência direita.
- Vai fazendo isso até encontrar, percorrendo um lado da árvore.



Implementação em Javascript

- É muito difícil.
- Consultar a matéria Análise de Algoritmos e Estrutura de Dados nas minhas anotações.