O que é busca binária?

"A busca binária é uma técnica eficiente para localizar um elemento em uma lista ordenada. Ao invés de procurar item por item, ela divide o espaço de busca ao meio a cada passo."

Estrutura do Código

a) Importações:

- *fs* para leitura de arquivo.

A função *fs* vem do módulo nativo do Node.js chamado *fs **(file system)**,* que serve para **ler, escrever, atualizar e deletar arquivos** no sistema de arquivos.

- *readline* para entrada do usuário.

A função *readline* serve para **ler entradas do usuário pelo terminal**, linha por linha. É muito útil para criar **interfaces de linha de comando interativas**, como esse programa que pede uma palavra para buscar.

b) lerArquivo

Tem como objetivo **ler o conteúdo de um arquivo de texto** e transformá-lo em um **array de palavras**, onde cada palavra está separada por linha no arquivo. Abaixo está uma explicação passo a passo:

Definição da Função:

function lerArquivo(arquivo: string): string[] {

- A função recebe como parâmetro o nome (ou caminho) do arquivo (arquivo: string).
- Ela retorna um array de strings (string[]), onde cada string é uma palavra do arquivo.

Leitura do Conteúdo do Arquivo:

const conteudo = fs.readFileSync(arquivo, 'utf-8');

- Usa *fs.readFileSync* (leitura síncrona) para abrir e ler todo o conteúdo do arquivo.
- O segundo parâmetro 'utf-8' garante que o conteúdo seja lido como texto (e não como binário).

Divisão do conteúdo em linhas

const palavras = conteudo.split(/\r?\n/).filter(Boolean);

- *.split(/\r?\n/):* divide o conteúdo onde houver quebra de linha (\n em Linux/macOS ou \r\n em Windows).
- *.filter(Boolean):* remove quaisquer linhas vazias (por exemplo, se o arquivo terminar com uma quebra de linha extra).

Retorno

return palavras;

- Retorna o array com todas as palavras (uma por linha no arquivo original).

Resumo

A função *lerArquivo*:

- Abre um arquivo de texto.
- Lê todo o conteúdo de uma vez.
- Divide esse conteúdo em um array de linhas.
- Remove linhas vazias.
- Retorna esse array de palavras para ser usado na busca binária.

c) buscaBinaria

- Explique o uso de *localeCompare* para comparação de strings com acento ou maiúsculas/minúsculas (e que a ordenação deve ser consistente com essa função).

Definição da função

function buscaBinaria(palavras: string[], alvo: string): { posicao: number, passos: number }

- **Parâmetros**:
  - palavras: um array de strings (que deve estar em ordem alfabética).
  - alvo: a palavra que você quer encontrar.
- **Retorno**:
  - Um objeto com duas propriedades:
    - posicao: o índice onde a palavra foi encontrada, ou -1 se não encontrada.
    - passos: quantas comparações foram feitas até encontrar (ou não) o valor.

Inicialização de variáveis

```
let inicio = 0;
let fim = palavras.length - 1;
let passos = 0;
```

- Define o intervalo inicial da busca (inicio e fim).
- passos conta quantas vezes o laço executa (quantidade de comparações feitas).

Laço da busca binária

```
while (inicio <= fim) {
  passos++;
  const meio = Math.floor((inicio + fim) / 2);
  const comparacao = palavras[meio].localeCompare(alvo);
```

- Enquanto o intervalo ainda for válido (inicio <= fim):
  - Incrementa passos.
  - Calcula o índice meio (ponto central do intervalo).
  - Compara a palavra no meio com a palavra buscada usando localeCompare.

Resultado da comparação

```
if (comparacao === 0) {
  return { posicao: meio, passos };
}
else if (comparacao < 0) {
  inicio = meio + 1;
}
else {
  fim = meio - 1;
}
```

- Se localeCompare retornar 0, as palavras são iguais → retorna posição e passos.
- Se a palavra no meio for **menor** que o alvo → busca na metade **direita**.
- Se for **maior** → busca na metade **esquerda**.

Se não encontrar

return { posicao: -1, passos };

- Se o laço terminar e a palavra não for encontrada, retorna -1 e o número de passos.

Resumo

A função buscaBinaria:

- Executa uma **busca eficiente** num vetor ordenado de palavras.
- Divide o intervalo pela metade a cada passo.
- Retorna a **posição da palavra** (ou -1) e o **número de comparações** feitas.

d) main

Definição da função

function main() {

Essa é a função principal do programa. Ela coordena toda a execução: leitura do arquivo, entrada do usuário e busca binária.

Criação da interface de entrada

```
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
```

- Usa o módulo readline para criar uma **interface de leitura e escrita via terminal**.
- input: process.stdin → lê o que o usuário digita.
- output: process.stdout → escreve mensagens no terminal.

Solicita o nome do arquivo ao usuário

rl.question('Digite o nome do arquivo: ', (arquivo) => {

- Pergunta ao usuário qual arquivo ele deseja usar (ex: palavras.txt).
- O valor digitado será passado como argumento para a função callback e armazenado na variável arquivo.

Verifica se o arquivo existe

```
if (!fs.existsSync(arquivo)) {
  console.log('Arquivo não encontrado.');
  rl.close();
  return;
}
```

- Usa fs.existsSync() para checar se o arquivo realmente existe.
- Se não existir:
    - Mostra uma mensagem de erro.
    - Encerra a interface (rl.close()) e termina a execução com return.

Lê o conteúdo do arquivo

const palavras = lerArquivo(arquivo);

- Chama a função lerArquivo() passando o nome do arquivo.
- Armazena as palavras (em array de strings) na variável palavras.

Pergunta qual palavra o usuário quer buscar

rl.question('Digite a palavra a ser buscada: ', (palavra) => {

- Solicita ao usuário qual palavra ele quer procurar no arquivo carregado.

Realiza a busca binária

const resultado = buscaBinaria(palavras, palavra);

- Chama a função buscaBinaria() com a lista de palavras e a palavra alvo.
- Armazena o resultado (posição + número de passos).

Mostra os resultados da busca

```
if (resultado.posicao !== -1) {
  console.log(`Palavra encontrada na posição ${resultado.posicao}.`);
} else {
  console.log('Palavra não encontrada.');
}
console.log(`Número de passos de comparação: ${resultado.passos}`);
```

- Se a posição for diferente de -1, imprime a posição da palavra.
- Caso contrário, informa que a palavra não foi encontrada.
- Sempre exibe o número de comparações realizadas durante a busca.

Encerra a interface

rl.close();

- Finaliza a leitura do terminal (evita que o programa fique aberto esperando nova entrada).

Execução automática

- No final do script, chama a função main() para iniciar a execução do programa.

Resumo da main()

A função:

1. Pede o nome do arquivo.
2. Verifica se o arquivo existe.
3. Lê as palavras do arquivo.
4. Pede a palavra a ser buscada.
5. Realiza a busca binária.
6. Mostra os resultados.
7. Fecha a interface do terminal.

Possíveis perguntas de um professor

Lógica e Complexidade

“O que acontece se o arquivo não estiver em ordem alfabética?”

O algoritmo pode falhar ou retornar resultados incorretos, pois a busca binária depende da ordenação. Nesse caso, seria necessário ordenar primeiro.

“Qual a complexidade do seu algoritmo?”

O tempo de execução é O(log n) no pior caso, e O(1) no melhor caso. Já a leitura do arquivo é O(n), pois precisamos carregar todas as palavras.

“Por que você usou localeCompare **em vez de** ==**?”**

Porque localeCompare trata melhor diferenças de acento, maiúsculas/minúsculas, e segue a ordenação da língua local, o que é ideal para textos em português.

“Você consegue adaptar o programa para aceitar palavras fora de ordem e ainda assim funcionar?”

Sim, bastaria adicionar uma etapa de ordenação no vetor após a leitura do arquivo usando .sort().

Código e Implementação

“Por que você usou readFileSync **e não** readFile **assíncrona?”**

Como o programa é simples e linear, a versão síncrona facilita a leitura e controle do fluxo. Em sistemas maiores, a versão assíncrona é mais indicada para evitar bloqueios.

“Como você lidaria com palavras com acento ou caixa alta?”

Uma opção é normalizar todas as palavras com .toLowerCase() e normalize('NFD').replace(/\p{Diacritic}/gu, "") antes da busca, para garantir igualdade.

Reflexões e Extensões

“Como você mostraria visualmente a eficiência da busca binária?”

Poderia comparar os passos da busca binária com os da busca linear em uma lista grande e exibir gráficos com n versus número de passos.

“Como adaptaria para buscar múltiplas palavras ao mesmo tempo?”

Poderia ler uma lista de palavras a serem buscadas e usar um for para aplicar a busca binária a cada uma delas, acumulando resultados.