**Introdução**

Os cientistas da computação estão sempre em busca de métodos de otimização, a equação de usar menos recursos, mas com resultados efetivos e de respostas rápidas. Dadas as necessidades é necessário reproduzir, comparar e simular as ṕossíveis soluções de problemas . Este relatório da disciplina de Estrutura de dados básicas I tem o intuito de detalhar o processo de Análise empírica de algoritmos, específicamente algoritmos de busca.

Passando do limiar teórico a construção lógica de uma solução a sua aplicação em um cenário real demanda limites de recursos físicos como o espaço de memória e temporal, códigos que demorem muito para finalizar sua tarefa são totalmente indesejados. Tratando-se do mundo atual em que lidamos com uma quantidade gigantesca de dados qualquer otimização de código é bem vinda e a comparação de resultados permite a visualização do comportamento em um cenário estressante com uma entrada gigante. Um busca linear simples pode não fazer muita diferença para busca em um vetor de duzento, quinhentos, *n* elementos, mas não é a melhor escolha para um vetor de um milhão de itens, por exemplo.

Os métodos testados neste trabalho foram: **busca linear simples e recursiva**, busca binária simples e recursiva, busca ternária, busca fibonacci e jump search. Os resultados levaram a uma série de hipóteses sobre o porquê do seu respectivo resultado gráfico.

**metodologia**

O material principal e que foi produzido para a realização desse relatório encontra-se disponível no github no link <https://github.com/MoisesMsa/analiseEmpirica>**.** O projeto foi elaborado em C++ versão 11, conta com CMake para sua compilação.

O código foi executado em uma máquina com processador core i3, com 4GB de RAM, e placa mãe \_\_\_\_\_\_\_ cujo modelo é o lenovo ideapad320.

A compilação é feita através do g++ versão xx.xx.

No repositório estão disponíveis os códigos **busca linear simples e recursiva**, busca binária simples e recursiva, busca ternária, busca fibonacci e jump search.

//parágrafo muito grande dividir

Ao compilar e iniciar o programa os algoritmos selecionados são executados **após a alocação do vetor de buscas com a quantidade de elementos definida pelo usuário (ou padrão caso não seja solicitada) maior ou igual ao total da máquina utilizada para os testes X milhões de itens**, durante seu tempo de execução a biblioteca ***std::chronos*** faz a contagem do intervalo de tempo até a chegada ao fim do vetor (tempo final - tempo inicial) através da função ***chronos::clock****,* o resultado final é armazenado em um buffer e ao final todo o conteúdo guardado é passado para um arquivo “tempos.txt”. O**s tempos de execução são resultantes da média de 100 execução no intervalo definido pelo tamanho máximo do vetor de busca**. Com a geração do arquivo os dados são plotados com a ajuda da biblioteca **gnoplot** em um gráfico **que relaciona o tempo (x) com o quantidade de itens (y).**

**A corretude do que foi implementada foi testada com o uso da biblioteca googleTests e com os teste de corretude.**

Por se tratar de uma análise em situações estressantes, ou seja, o pior caso possível de acordo com as limitações do hardware as buscas foram executados com X número de testes com um total de X itens em um vetor para a consulta do valor procurado que não pertencente a esse conjunto.

**A literatura utilizada para discussão dos resultados foi basicamente \_\_\_\_\_\_\_\_\_\_\_\_.**

Resultados

**Busca linear**

o problema da busca linear “dado um vetor de tamanho N e uma chave k”

complexidade

número de passos

invariante de laço da busca

melhor caso

pior caso

solução definida e implementada:

//pseudo código aqui

imagem 1 - gráfico da busca linear

No gráfico é possível visualizar o crescimento em relação tempo. Há uma longa variação

**Discussão**

pq é melhor? a variação foi grande a partir do que? a partição? recursão vs. iterativa.