UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
CAMPUS NATAL CENTRAL
CURSO DE BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

ANÁLISE EMPÍRICA DOS ALGORITMOS DE BUSCA

Tobias dos Santos Neto
Wisla Alves Argolo

Natal-RN
Abril, 2023

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
CAMPUS NATAL CENTRAL
CURSO DE BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

ANÁLISE EMPÍRICA DOS ALGORITMOS DE BUSCA

Tobias dos Santos Neto
Wisla Alves Argolo

Relatório técnico do desempenho dos algoritmos de busca estudados na Unidade 1 da disciplina de Estruturas de Dados Básicas I, como requisito parcial para obtenção de nota desta unidade.

Natal-RN
Abril, 2023

Sumário

1 Introdução

Um algoritmo consiste em um conjunto de instruções organizadas sistematicamente para resolver um problema computacional (SZWARCFITER; MARKENZON, 2015). Apesar de constantemente ser possível solucionar o mesmo problema a partir de múltiplos algoritmos, estes possuem eficiências distintas, isto é, apresentam tempo de execução e uso de memória diferentes. A relevância dessa diferença está relacionada com o volume de dados, aumentando à medida que este cresce (DROZDEK, 2001).

Nesse sentido, a análise da eficiência de algoritmos tem um papel fundamental na área da computação, bem como a medição do tempo de execução destes. Esse tempo pode ser obtido através de métodos empíricos, que consistem em executar o algoritmo e medir o tempo para entradas de tamanhos distintos, ou por meio de análise matemática (SZWARCFITER; MARKENZON, 2015; SEDGEWICK, 1998).

Pensando nisso, este relatório propõe a análise empírica de 3 algoritmos de busca distintos para diferentes volumes de dados de entrada. O objetivo desse processo consiste em examinar a velocidade dos algoritmos de busca para cada uma das a fim de determinar o mais eficiente em termos de tempo de execução, de modo a auxiliar o programador no planejamento futuro e aplicação correta de cada algoritmo.

Para permitir a análise dos algoritmos, o trabalho está dividido em seções que apresentarão a introdução e contextualização do problema apresentado, os materiais e métodos utilizados, os resultados obtidos pela implementação dos algoritmos e as análises e conclusões geradas a partir de tais dados.

2 Metodologia

2.1 Materiais utilizados

2.1.1 Computador e Sistema Operacional

O computador empregado foi Lenovo Ideapad S145 com processador AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz e RAM 12 GB.

Em relação ao sistema operacional, foi utilizado o Windows 11 Home Single Language versão 22H2 e o terminal WSL (*Windows Subsystem for Linux*) executando a versão 20.04 da distribuição Linux Ubuntu.

2.1.2 Ferramentas de programação

Os 3 algoritmos analisados foram implementados na linguagem C++17 a partir do editor de código-fonte gratuito e de código aberto VS Code (*Visual Studio Code*).

Além disso, tais algoritmos foram compilados no WSL utilizando o cmake versão 3.26.0 a partir dos seguintes comandos:

cmake -S source -B build -D CMAKE_BUILD_TYPE=Release
cmake –build build (aciona o processo de compilação dentro da pasta build)

Em que *source* corresponde pasta na qual o cmake será encontrado.

Para as medições de tempo, foi utilizada a biblioteca *chrono*. Já a biblioteca *fstream* foi responsável pela escrita dos dados em arquivos de extensão *.txt*.

2.1.3 Gráficos

Os gráficos obtidos neste trabalho foram gerados através do aplicativo de visualização Gnuplot versão 5.4 *patchlevel* 6 para Windows.

2.1.4 Algoritmos

2.1.4.1 Busca linear

Este algoritmo consiste em analisar cada elemento do conjunto, desde o primeiro até encontrar o elemento procurado ou determinar que o elemento não está presente no intervalo

Algorithm 1: Busca linear

Input : Elemento $value$ procurado, ponteiro para o primeiro elemento do intervalo de busca $first$ (incluso) e ponteiro para último elemento do intervalo de busca $last$ (não incluso)

Output : Ponteiro para o elemento encontrado; ou $last$ caso o elemento não seja encontrado

Function `lsearch` (*value_type *first, value_type *last, value_type value*) **:**

```
for int * ptr ← first to last − 1 do
    if *ptr = value then
        return ptr;
    end
end
return last;
```

2.1.4.2 Busca binária

Esta solução verifica se o intervalo de pesquisa possui o elemento procurado e a cada comparação reduz o intervalo de pesquisa pela metade.

Algorithm 2: Busca binária

Input : Elemento $value$ procurado, ponteiro para o primeiro elemento do intervalo de busca $first$ (incluso) e ponteiro para último elemento do intervalo de busca $last$ (não incluso)

Output : Ponteiro para o elemento encontrado; ou $last$ caso o elemento não seja encontrado

Function `bsearch` (*value_type *first, value_type *last, value_type value*) **:**

```
value_type * aux ← last;
while first ≠ last do
    value_type * mid ← first + (last − first)/2;
    if value = *mid then
        return mid;
    end
    if value > *mid then
        first ← mid + 1;
    end
    else
        last ← mid;
    end
end
return aux;
```

2.1.4.3 Busca binária recursiva

Esta solução realiza processo semelhante a busca binária, mas utilizando recursão.

Algorithm 4: Busca binária recursiva

Input: Elemento $value$ procurado, ponteiro para o primeiro elemento do intervalo de busca $first$ (incluso) e ponteiro para último elemento do intervalo de busca $last$ (não incluso)

Output: Ponteiro para o elemento encontrado; ou $nullptr$ caso o elemento não seja encontrado

Function `aux_rec_bsearch` *(value_type *first, value_type *last, value_type value)*:

$value\_type * mid \leftarrow first + (last - first)/2$;
if $value = *mid$ **then**
  return mid;
end
while $first \neq last$ **do**
  if $value > *mid$ **then**
    $first \leftarrow mid + 1$;
    return $aux\_rec\_bsearch(first, last, value)$;
  end
  else
    $last \leftarrow mid$;
    return $aux\_rec\_bsearch(first, last, value)$;
  end
end
return $nullptr$;

Input: Elemento $value$ procurado, ponteiro para o primeiro elemento do intervalo de busca $first$ (incluso) e ponteiro para último elemento do intervalo de busca $last$ (não incluso)

Output : Ponteiro para o elemento encontrado; ou $last$ caso o elemento não seja encontrado

Function `bsearch_rec` *(value_type *first, value_type *last, value_type value)* **:**

$value\_type * aux \leftarrow aux\_rec\_bsearch(first, last, value)$;
return $aux \ ? \ aux : last$;

2.2 Obtenção e tratamento dos dados

Neste trabalho, foi utilizada uma análise empírica dos algoritmos de busca. Para tanto, cada algoritmo foi testado em relação ao pior caso, isto é, na busca de um elemento que não está presente no intervalo de pesquisa. Além disso, os algoritmos foram executados com uma entrada inicial de um array com 10^3 elementos e posteriormente foram executados com entradas crescentes de até 10^7 elementos, adicionando 20 mil elementos e medindo o tempo de execução (em

milissegundos) a cada novo teste. Esse processo foi realizado 10 vezes para cada implementação.

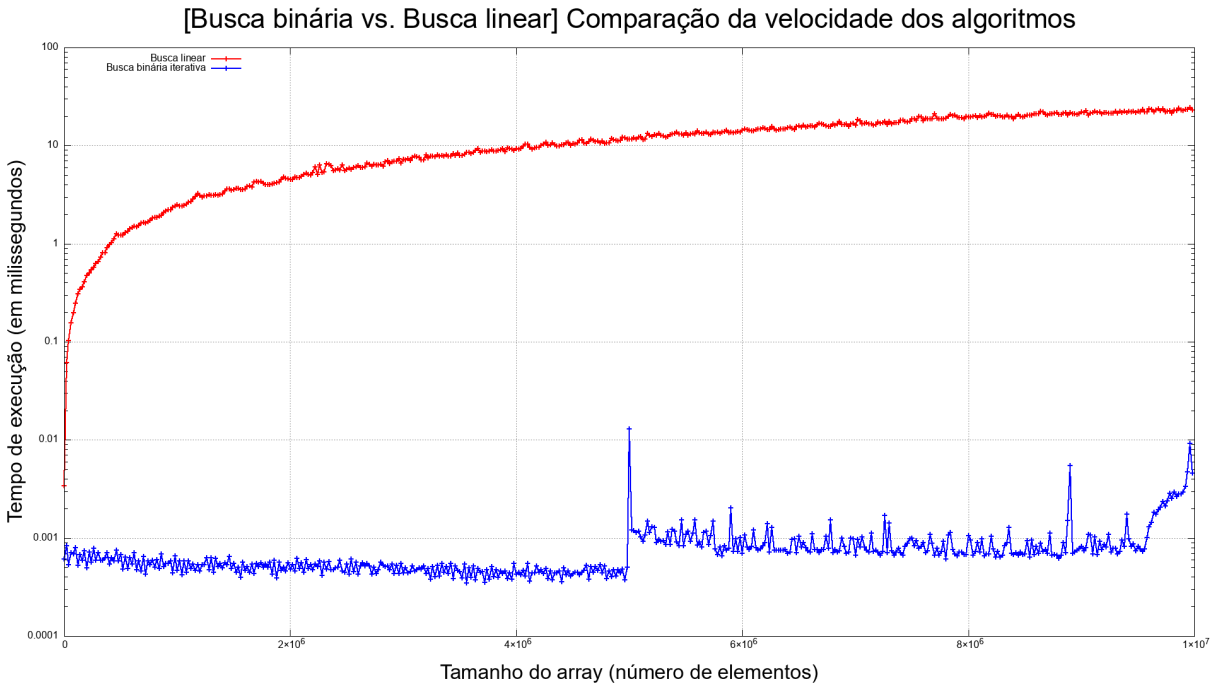
A partir desses dados, 10 tabelas em arquivo *.txt* foram geradas para cada algoritmo, relacionando o tamanho da entrada e o tempo de execução. Disso, as médias aritméticas dos tempos de execução das 10 tabelas para cada entrada foram calculadas e inseridas em um novo arquivo *.txt*. Por fim, com base nos arquivos, foram plotados gráficos, em escala logarítmica no eixo y, para visualizar os resultados de forma resumida e permitir a comparação do desempenho entre os diferentes algoritmos de busca estudados.

3 Resultados e discussão

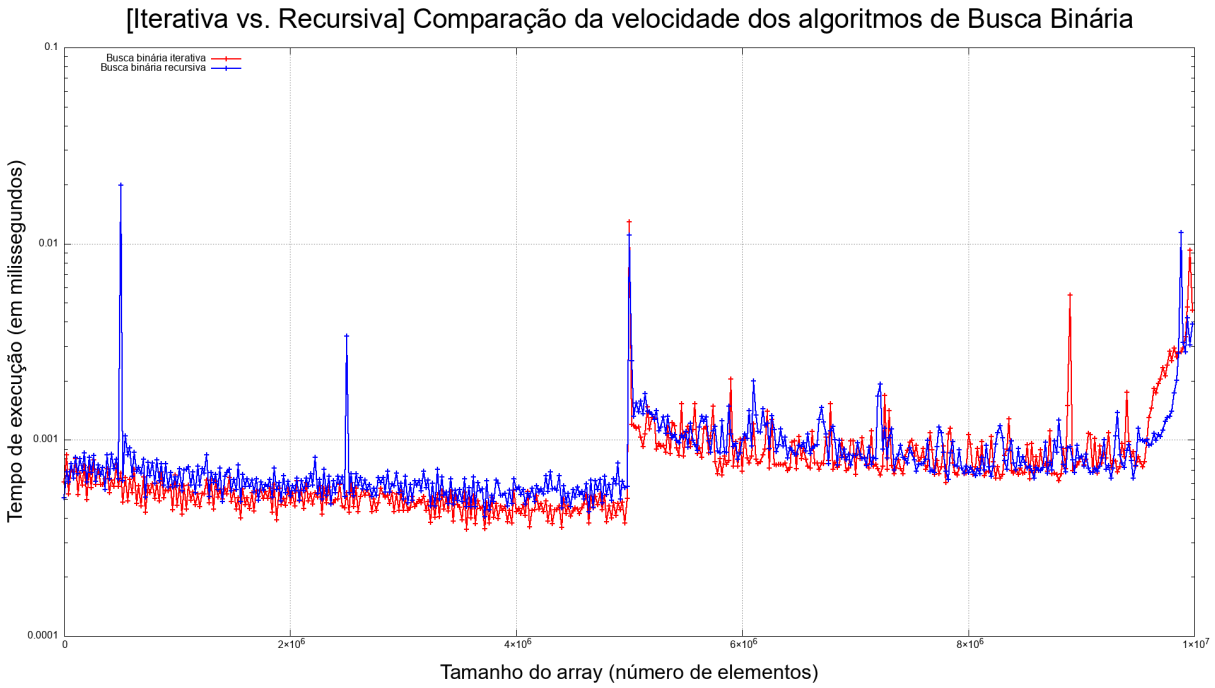
Nesta seção, apresentaremos os gráficos, em escala logarítimica no eixo y, que relacionam as diferentes entradas (tamanho da amostra) e o tempo de execução (em milissegundos) no pior caso para cada algoritmo, assim como as análises resultantes.

3.1 Resultados

3.1.1 Busca linear vs. Busca binária



3.1.2 Busca binária recursiva vs. Busca binária iterativa



3.2 Discussão

A fim de alcançar uma precisão maior no estudo da eficiência dos algoritmos, os testes foram realizados dez vezes e a média do tempo de execução dos testes foi utilizado para construir os gráficos dispostos acima.

A primeira análise foi feita comparando a busca linear com a busca binária e o gráfico 1 evidencia a diferença entre o tempo de execução das buscas. Como o esperado, a busca binária com seu tempo de execução menor mostrou-se mais eficiente para todos os tamanhos de array testados. Enquanto a busca linear, em seu pior caso, realiza N comparações (onde N é o tamanho do array), a busca binária realiza aproximadamente $1 + \lfloor \log_2 N \rfloor$ (isto é, uma unidade a mais que a parte inteira do valor $\log_2 N$), portanto, o resultado dos dados foram consistentes com o estudo da complexidade destes algoritmos.

Diferente da primeira comparação, em que houve uma análise à respeito da diferença do tempo de execução entre dois tipos de busca distintos, no segundo caso é avaliado a diferença entre o tempo de execução de buscas binárias (mesmo tipo), mas implementadas de formas diferentes: uma recursiva e outra iterativa.

Nesse contexto, o segundo gráfico sugere uma eficiência semelhante, se não equivalente, entre as buscas binárias recursiva e iterativa, uma vez que a diferença entre o tempo de execução das buscas é mínima. É importante ressaltar que para tamanhos menores de array, a busca binária iterativa mostrou-se relativamente melhor que a busca binária recursiva, enquanto para tamanhos próximos a $5 \cdot 10^6$ ambos algoritmos sofrem um aumento significativo no tempo de execução e a

diferença entre estes cai bastante. Dessa forma, pela pequena diferença dos tempos de execução ao longo do gráfico, é possível concluir que ambos algoritmos possuem a mesma eficiência.

4 Conclusão

A análise empírica é um método bastante utilizado quando é necessário verificar a eficiência de algoritmos para que seja possível compará-los e aplicá-los de modo adequado na resolução de problemas. Com base nisso, os seguintes algoritmos de busca foram examinados empiricamente quanto ao seu tempo de execução: i) busca linear; ii) busca binária iterativa e iii) busca binária recursiva.

Nesse contexto, os resultados obtidos evidenciaram que a busca binária é mais apropriada para lidar com grandes conjuntos de dados em comparação com a busca linear. Além disso, a busca binária iterativa e a busca binária recursiva apresentaram tempos de execução similares para as diferentes entradas e, portanto, apresentaram velocidades semelhantes.

À vista disso, o desenvolvimento e resultados do trabalho mostraram-se bastante satisfatório, dado que atenderam os objetivos propostos.

Referências

DROZDEK, A. **Data structures and algorithms in C++**. [S.l.]: Brooks/Cole, 2001. 664 p. ISBN 0 534-37597-9.

SEDGEWICK, R. **Algorithms in C**. [S.l.]: Addison-Wesley Publishing Company, 1998. 720 p. ISBN 0-201-31452-5.

SZWARCFITER, J. L. S.; MARKENZON, L. **Estruturas de Dados e Seus Algoritmos**. Rio de Janeiro: LTC — Livros Técnicos e Científicos Editora Ltda, 2015. 236 p. ISBN 978-85-216-2994-8.