UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
FERNANDO FERREIRA DE LIMA FILHO

ANÁLISE EMPÍRICA DE ALGORITMOS DE BUSCA

NATAL - RN
Setembro de 2020

FERNANDO FERREIRA DE LIMA FILHO

ANÁLISE EMPÍRICA DE ALGORITMOS

Trabalho apresentado à Universidade Federal do Rio Grande do Norte como parte para a aprovação na disciplina de Estruturas de Dados I, ministrada pelo professor Selan.

NATAL - RN
Setembro de 2020

SUMÁRIO

1.INTRODUÇÃO

Um **algoritmo** é uma sequência finita de ações executáveis que visam obter uma solução para um determinado tipo de problema(Ziviani, 2011, p.). Algoritmos são procedimentos precisos, não ambíguos, mecânicos, eficientes e corretos(Dasgupta, Papadimitriou e Vazirani, 2010, p. 2). Neste trabalho foram implementados algoritmos de busca linear, binária iterativa e recursiva.

Na **busca linear** a ideia é expressar um tipo de pesquisa em vetores ou listas de modo sequencial, i. e., elemento por elemento, de modo que a função do tempo em relação ao número de elementos é linear.

Já a **busca binária** é um algoritmo de busca em vetores que segue o paradigma de divisão e conquista. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor.

A causa da implementação desses algoritmos foi a análise empiricamente o tempo de execução de cada, ou seja, queremos verificar quanto tempo demora para cada um executar o papel para o qual foi designado. Além disso outro motivo para isso é comparar os algoritmos entre si e verificar qual tem maior eficiência, para que desse modo o mesmo que escreve este relatório consiga discernir qual algoritmo usar levando em questão o contexto em que se encontra.

2.METODOLOGIA

O experimento tomou como parâmetros a quantidade da entrada de dados e o tempo de execução em milissegundos do algoritmo em cada entrada. O cenário de execução assumido foi quando ocorre o pior caso de cada algoritmo, ou seja, quando o elemento procurado não pertence ao vetor. Os dados foram vetores de valores inteiros ordenados com tamanho inicial de 100.000.000. Ao fim de cada execução a quantidade de inteiros foi aumentada em 18.000.000 e o algoritmo executado novamente, esse processo foi repetido até se completar uma quantidade de 1.000.000.000 elementos no vetor. A cada execução o tamanho do vetor foi gravado em um arquivo do tipo “.data” em conjunto com seu respectivo tempo de execução. Ao fim da coleta de dados foram gerados tabelas e gráficos no intuito de ajudar na análise e assim discernir qual tem uma maior eficiência.

Foram utilizadas as seguintes ferramentas durante a execução do experimento, editor de código VsCode, compilador gcc for x86_64-linux-gnu 9.3.0, gerador automatizado Cmake, debugger gdb, linguagem C++, sistema operacional Linux Ubuntu 20.04.

Foram utilizados as seguintes ferramentas pós execução do experimento, google docs, google planilha, gnuplot.

O processo foi realizado através de uma máquina ideapad S145, intel Core I5-8265U, 8GB de memória, armazenamento de 1TB.

3.RESULTADOS

Finalizado o processo de implementação dos algoritmos de busca linear (imagem 1), busca binária (imagem 2), busca binária recursiva (imagem 3). O tratamento dos dados consistiu em colocar em uma planilha o tamanho do vetor e o tempo de processamento de cada algoritmo, como pode ser visto na tabela 1.

```
value_typ  * lsearch( value_typ  * first, value_typ  * last, value_typ  value )
{                    e                   e                  e
    size_t size = last - first;
    for (size_t i = 0; i < size; i++)
    {
        if(*(first+i) == value) return (first+i);
    }
    return first+size;
}
```

Imagem 1: Algoritmo de busca linear

```
value_typ  * bsearch( value_typ  * first, value_typ  * last, value_typ  value )
{                     e                   e                  e
     value_typ * last_backup = last;
    while(first != last){
        size_t size = last - first;
        int middle = (size-1)/2;
        if(*(first+middle) == value) return (first+middle);
        else if(*(first+middle) < value) {
            first = (first+middle+1);
        }
        else {
            last = (first+middle);
        }
    }
    return last_backup;
}
```

Imagem 2: Algoritmo de busca binária iterativa

```
value_typ  * bsearch_recursiv ( value_typ * first, value_typ  * last, value_typ  value){
e   int middle { 0 };           e                 e                 e
    if(first != last){
        size_t size = last - first;
        if(*first > value) return last;
        if(*(first+middle) == value){
            return (first+middle);
        }
        if(*(first+middle) < value){
            return bsearch_recursiv ((first+middle+1), last,value);
        }          e
        return bsearch_recursiv (first, (first+middle), value);
        middle e (size-1)/2;

    }
    return last;
}
```

Imagem 3: Algoritmo de busca binária recursiva

Vetores	Linear	Binária iterativa	Binária recursiva
100000000	223.514 ms	0.001668 ms	4.8e-05 ms
208000000	485.853 ms	0.000946 ms	2.3e-05 ms
316000000	717.286 ms	0.000837 ms	2.3e-05 ms
406000000	895.756 ms	0.000835 ms	2.3e-05 ms
514000000	1139.56 ms	0.000860 ms	2.4e-05 ms
604000000	1334.07 ms	0.000823 ms	2.3e-05 ms
712000000	1577.12 ms	0.000855 ms	2.3e-05 ms
802000000	1768.68 ms	0.000983 ms	2.3e-05 ms
910000000	2009.51 ms	0.000829 ms	2.2e-05 ms
1000000000	2227.94 ms	0.000938 ms	2.2e-05 ms

Tabela 1: Tempo de execução para cada algoritmo

Como pode ser visto na tabela 1(acima) só foi posto os valores do vetor com diferencial de 100.000.000 entre cada , embora tenha sido executado um incremento de 18.000.000 elementos a cada nova execução, essa escolha foi feita para não estender demais a tabela, e também por conta que o tempo só mostrou diferença significativa com tais valores.

Com os dados coletados foram se gerados os seguintes gráficos.

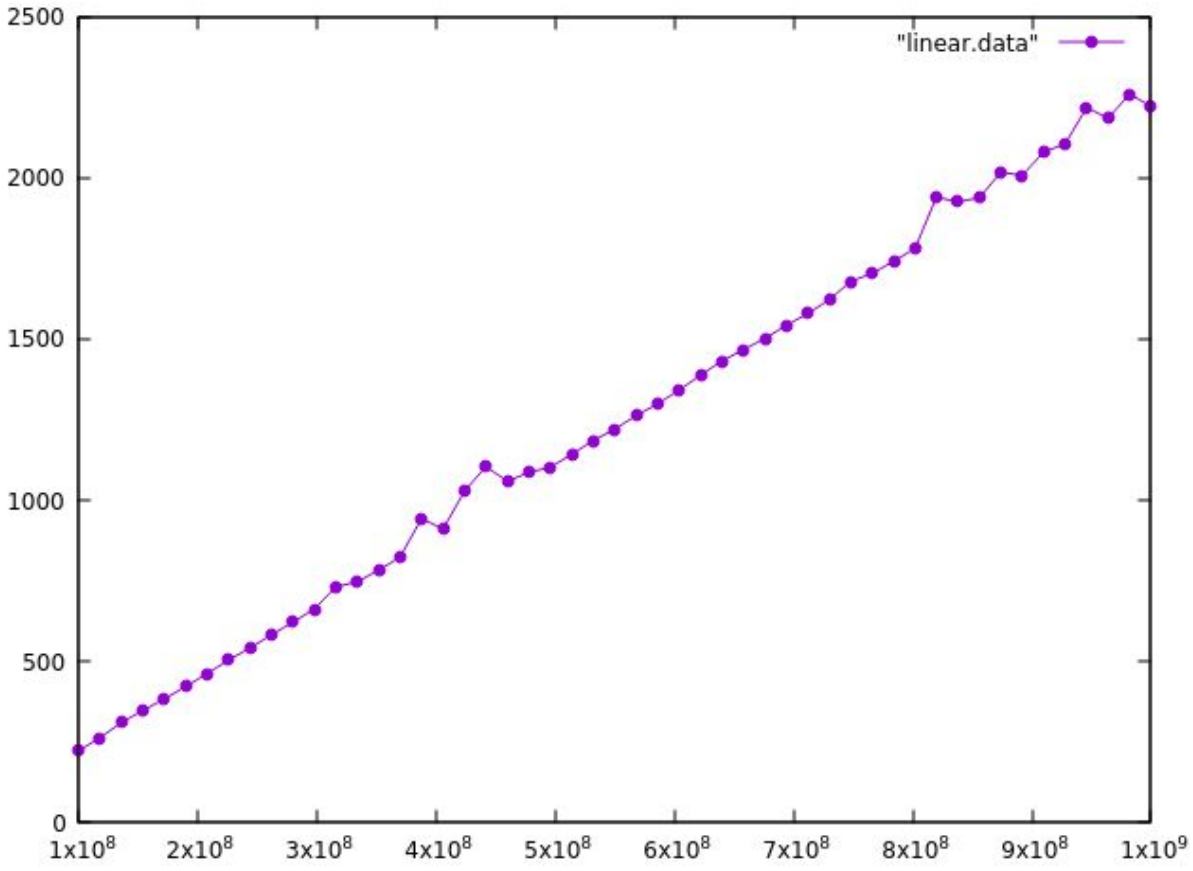

Gráfico 1:Busca Linear

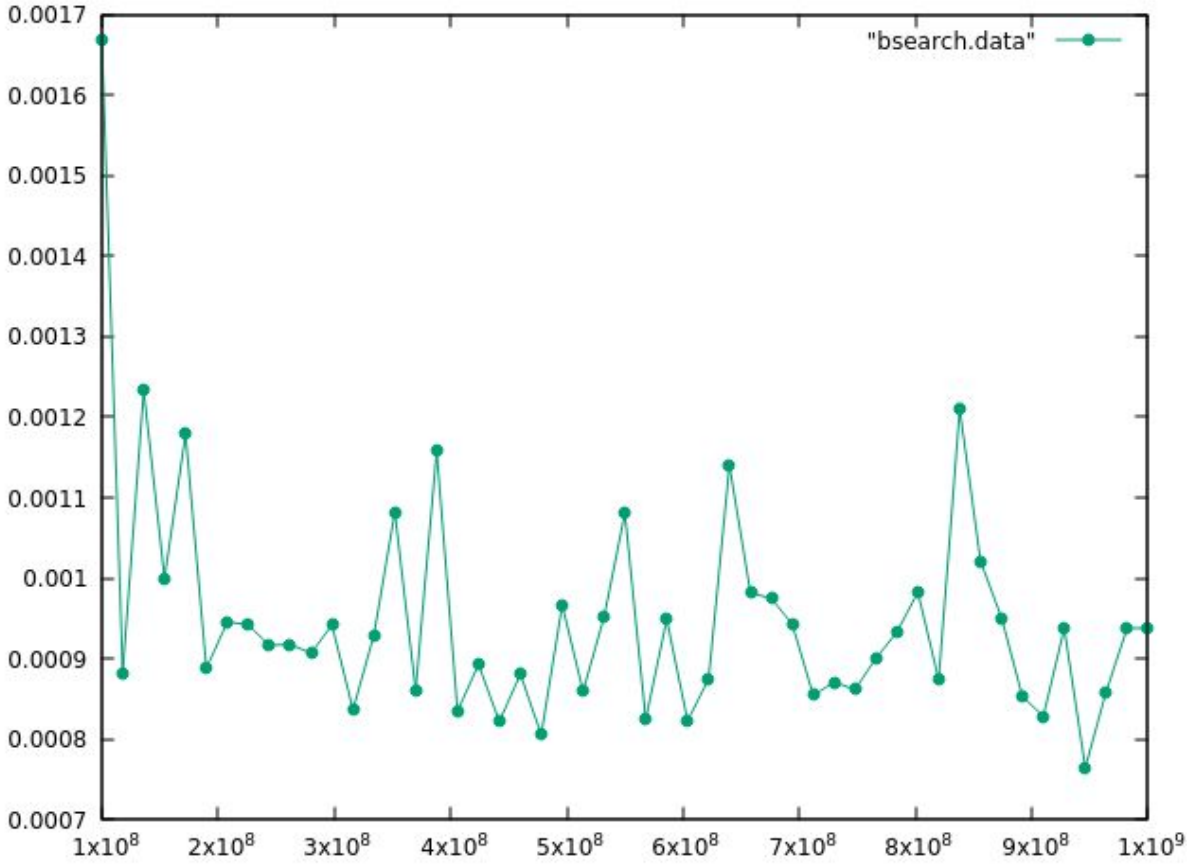

Gráfico 2: Busca binária iterativa

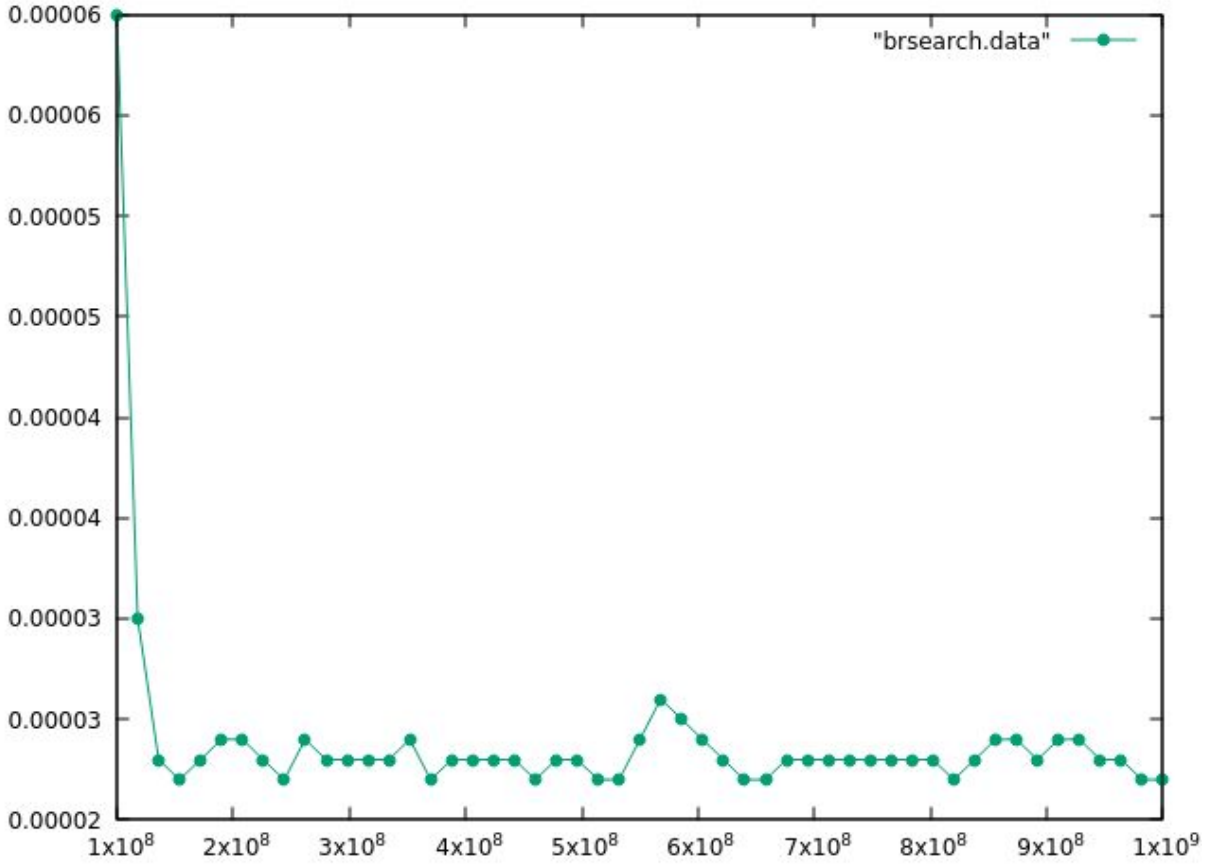

Gráfico 3: busca binária recursiva

4.ANÁLISE DOS RESULTADOS E CONCLUSÃO

Com a elaboração das planilhas e gráficos realizados demos início a nossa análise. Ela consistiu em comparar o tempo de execução da busca linear com a binária, também foi se comparada a busca binária iterativa com a recursiva , ao fim das comparações foram apresentadas as razões pelas quais isso ocorre.

Ao analisar o gráfico 1 e 2 vemos uma diferença evidente entre os algoritmos devido ao cenário assumido, ou seja, com grandes quantidades de dados para serem processados e com o elemento procurado não existente no vetor. Por conta disso quando observado o comportamento da busca linear ela terá que percorrer todo o vetor para concluir que o elemento não está presente. Já a busca binária não necessita fazer o mesmo, em sua primeira iteração ela irá descartar metade dos elementos existentes no vetor, pois se trata de um algoritmo de divisão e conquista, esse comportamento irá se repetir até o fim da execução da função. Por conta disso a busca binária se mostra mais eficiente que a linear.

Já ao analisar a função binária iterativa e recursiva somente com o recurso do gráfico fica difícil de saber qual é mais eficiente pois como pode ser visto nos gráficos 2 e 3 os tempos de execução se aproximam bastante, desse modo não utilizamos unicamente do recurso dos gráficos para concluir qual é mais eficiente. Logo para chegar uma conclusão de qual é mais eficiente foi se analisado o espaço de memória utilizado por cada um deles, funções recursivas exigem mais espaço de memória, devido a cada execução um novo espaço de memória deve ser reservado para execução de uma nova função, concluímos que a iterativa é mais eficiente do que a recursiva.

Desse modo fica demonstrado que a busca binária iterativa é mais eficiente do que a linear e a binária recursiva. Além disso fim desse trabalho fica claro que ao utilizar uma qualquer função é necessário analisar os possíveis cenários em que ela pode se encontrar e assim discernir qual irá ter um melhor desempenho.

5.REFERÊNCIAS

1.The Editors Of Encyclopaedia Britannica. Algorithm.

2.learn Cpp(https://www.learncpp.com/).

3. Cplusplus(https://www.cplusplus.com/).

4.Ziviani, 2011, p. 1.

5.Uma abordagem para avaliar o desempenho de algoritmos baseada em simulações automáticas de modelos de redes de petri coloridas Hierárquicas - UFU.