Atividade 2 - Algoritmos de Busca

Diogo do N. Paza[1]

[1]Universidade Estadual do Oeste do Paraná (UNIOESTE)
R. Universitária, 1619 - Universitário - 85819-110 - 4 Cascavel - PR - Brasil

diogopazacvel@gmail.com

Resumo. *Esta atividade consiste em implementar e comparar empiricamente a eficiência dos métodos de Busca Sequencial Padrão, Busca sequencial Otimizada Para Arranjos Ordenados, Busca Por Saltos (jump search) e Busca Binária. Para realização e comparação serão gerados diferentes cenários de teste aleatórios usando a linguagem de programção C.*

1. Introdução

O objetivo principal do trabalho é realizar a busca por determinados valores em um conjunto de dados e retornar o tempo de execução nos seguintes métodos de busca: Busca sequencial padrão, Busca sequencial otimizada para arranjos ordenados, Bucsa por saltos (jump search). Posteriormente com a linguagem R analisar, visualizar e manipular os dados gerados atraveś das buscas. A linguagem C será adotada como padrão. Em [Git] está disponível o código fonte, tabelas, planilhas e arquivos CSV (valore separados por vírguls) usados neste trabalho.

A operação de busca visa responder se um determinado valor está ou não presente em um conjunto de elementos. O primeiro método de busca usado no trabalho é a Busca Sequencial, esse é a mais simples. Consiste na mera varedura serial, entrada por entrada, devolvendo-se o índice da entrada cuja chave for igual à chave fornecida como argumento de pesquisa ou retornando -1 em caso de não encontrado [IME]; o segundo algoritmo é a Busca sequencial otimizada para arranjos ordenados. Quando a tabela está ordenada (por exemplo, em ordem crescente dos elementos), uma melhoria pode ser feita. Se durante o processo de busca, encontrarmos um elemento que já é maior que x, não adianta continuar procurando, pois todos os outros serão também maiores [IME]; o próximo algoritmo é a Busca Binária baseado no paradigma dividir para conquistar. Essa verifica se a posição do meio do vetor contem x, se x for menor do que o elemento do meio, então x deve estar na metade inferior do vetor. Se x for maior do que o elemento do meio, então x deve estar na metade superior do vetor. O processo se repete sempre com o elemento do meio da lista, a cada comparação a lista é dividida em duas, reduzindo o tempo de busca [IME]; por último a Busca por Saltos na qual a ideia é verificar menos elementos do que na pesquisa linear, avançando por etapas fixas ou pulando alguns elementos em vez de pesquisar todos [Gee], quando encontrar um valor maior do que x, o algoritmo volta fazendo uma busca linear apenas no último setor em que estava realizando o salto.

2. Metodologia

Para os testes foi gerado um vetor randômico sem repetições utilizando a função *srand* a qual objetiva inicializar o gerador de números aleatórios com o valor da função *time(NULL)*. Este por sua vez, é calculado como sendo o total de segundos passados

desde 1 de janeiro de 1970 até a data atual. Desta forma, a cada execução o valor da ”semente”será diferente gerando números não repetidos no vetor. Após a criação do vetor randômico foi criado outro vetor com cem valores sorteados aleatoriamente baseados no vetor randômico inicial. Os mesmos vetores criados foram utilizados para cada algoritmo de busca. Para realizar a comparação foram gerados cenários de testes aleatórios variando-se o tamanho do arranjo de 2 elevado 10 a 2 elevado 20 elementos, executado cem buscas para cada cenário e assim obtido os valores de comparação.

Como é necessário retornar o tempo de execução em milisegundos das buscas no vetor será utlizada a bilbioteca *time.h* disponível na linguagem C junto com a função *clock*. Essa retorna o tempo de execução exato do momento em que ela foi chamada. Para encontrar o tempo de execução de um programa será usado ela duas vezes, uma para capturar o tempo inicial e outra para capturar o tempo final da execução. Calculando o tempo final menos tempo inicial é obtido o tempo de execução do programa em milissegundos, dividindo esse valor pelo *CLOCKS_PER_SEC* teremos este valor em segundos, pois esta constante tem o valor de 1000000. Para obter o valor em milissegundos, pode-se dividir o CLOCKS_PER_SEC por 1000.

Para cada teste foi gerado um arquivo *csv*, esses arquivos estão disponíveis em [Git]. Nesses arquivos foram criadas seis variaveis para análise: a primeira é a variável Busca que retorna o nome da busca; por segundo a variável Posicao que retorna a posição em que o elemento se encontra no vetor, caso o valor não exista é retornado -1; ValorPesquisado retorna o elemento que está sendo procurado no vetor; TempoResposta tráz o tempo decorrido para cada busca em milisegundos; TempoOrdenarVetor retorna o tempo gasto na ordenação do vetor; TempototalBusca é a soma do TempoResposta mais TempoOrdenarVetor quando o algoritmo faz uso de vetor ordenado.

A tabela 1 abaixo responde as seguintes questões de pesquisa: o valor foi encontrado nas 10 primeiras posições do arranjo (posição 0 a 9); o valor foi encontrado nas 10 últimas posições do arranjo (posição n-10 a n-1); o valor não foi encontrado no arranjo. Onde cada valor representa a média das buscas. Referente a questão a qual pergunta se o valor foi encontrado nas últimas 10 posições, apenas o vetor com tamanho de 8192 encontrou um valor em média. A média do tempo de busca dos valores encontrados nas 10 primeiras posições e o desvio padrão estão disponíveis em [Git].

Tabela 1. Questões de pesquisa.

Encontrado 10 Primeiras Posições	Valor Não Encontrado	Tamanho do Vetor
9	316	1024
6	360	2048
2	237	4096
6	260	8192
0	228	16384
2	208	32768
3	188	65536
0	244	131072
0	372	262144
0	376	524288
0	172	1048576

Tabela 2. A Tabela 2 mostra os valores médios e o desvio padrão do desempenho dos algoritmos

Algoritmo	Média Tempo de Execução	Desvio Padrão	Tempo Ordenação Vetor	Tamanho
Busca Sequencial	0.00664	0.001834132	Não-contém	1024
Busca Seq. Vetor Ordenado	0.00168	0.001496663	6.698000	1024
Busca Binária	0.00147	0.0005765624	6.698000	1024
Busca com Saltos	0.00184	0.00106097	6.698000	1024
Busca Sequencial	0.01332	0.00254209	Não-contém	2048
Busca Seq. Vetor Ordenado	0.00158	0.0005160064	26.074000	2048
Busca Binária	0.00148	0.0005021167	26.074000	2048
Busca com Saltos	0.00172	0.0005519479	26.074000	2048
Busca Sequencial	0.02393	0.005682251	Não-contém	4096
Busca Seq. Vetor Ordenado	0.00311	0.001090779	106.246000	4096
Busca Binária	0.00163	0.0005252224	106.246000	4096
Busca com Saltos	0.00207	0.002275273	106.246000	4096
Busca Sequencial	0.0407	0.01524514	Não-contém	8192
Busca Seq. Vetor Ordenado	0.00405	0.001677992	428.2	8192
Busca Binária	0.00156	0.0005563227	428.2	8192
Busca com Saltos	0.00186	0.0004025173	428.2	8192
Busca Sequencial	0.07689	0.03256909	Não-contém	16384
Busca Seq. Vetor Ordenado	0.00558	0.002370611	1756	16384
Busca Binária	0.00155	0.0005198096	1756	16384
Busca com Saltos	0.00189	0.0003993682	1756	16384
Busca Sequencial	0.1518	0.05959844	Não-contém	32768
Busca Seq. Vetor Ordenado	0.01290	0.007699534	7349	32768
Busca Binária	0.00171	0.001335566	7349	32768
Busca com Saltos	0.00224	0.001577109	7349	32768
Busca Sequencial	0.2757	0.1427198	Não-contém	65536
Busca Seq. Vetor Ordenado	0.02875	0.01860834	31041	65536
Busca Binária	0.00181	0.000419114	31041	65536
Busca com Saltos	0.00225	0.0005388915	31041	65536
Busca Sequencial	0.6117	0.2536822	Não-contém	131072
Busca Seq. Vetor Ordenado	0.02197	0.01211765	31041	131072
Busca Binária	0.00181	0.0004860685	119774	131072
Busca com Saltos	0.00221	0.0004333333	119774	131072
Busca Sequencial	1.524	0.2124594	Não-contém	262144
Busca Seq. Vetor Ordenado	0.01089	0.005128441	429792	262144
Busca Binária	0.00177	0.0004893822	429792	262144
Busca com Saltos	0.00209	0.000428646	429792	262144
Busca Sequencial	3.053	0.4057789	Não-contém	524288
Busca Seq. Vetor Ordenado	0.00990	0.004188729	1716354	524288
Busca Binária	0.00179	0.0004983812	1716354	524288
Busca com Saltos	0.00209	0.000446196	1716354	524288
Busca Sequencial	4.136	2.359406	Não-contém	1048576
Busca Seq. Vetor Ordenado	0.3053	0.1663446	8015753	1048576
Busca Binária	0.00229	0.001472269	8015752	1048576
Busca com Saltos	0.00302	0.0006663636	8015752	1048576

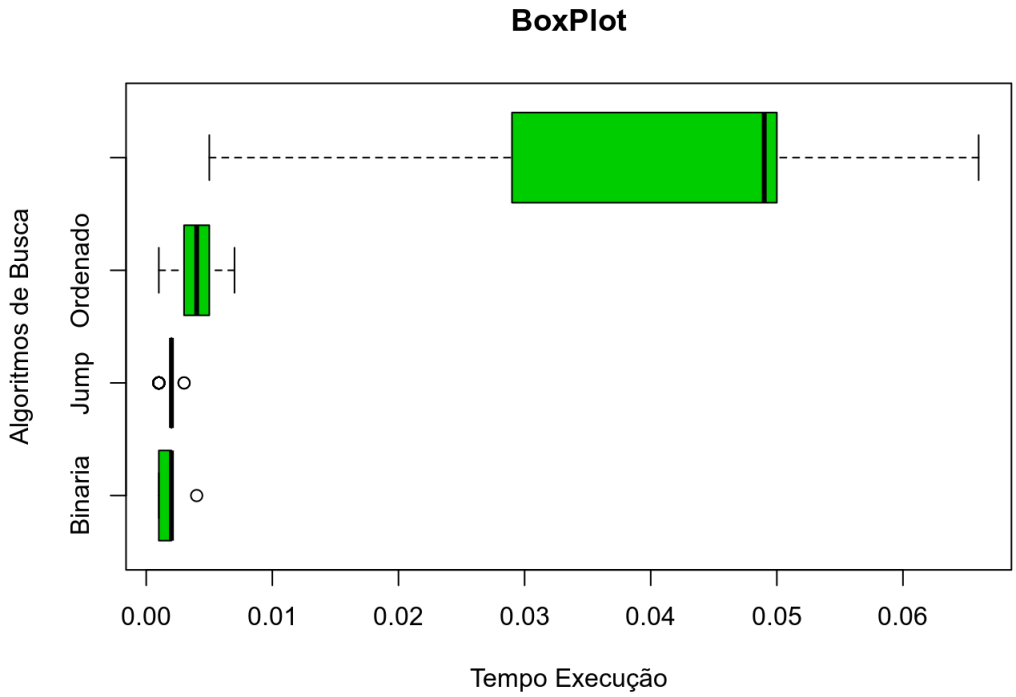

Figura 1. Gráfico de Boxplot do tempo de execução médio em milisegundos da Tabela 2.

A Tabela 2 traz a média das buscas, o desvio padrão e a média com o vetor ordenado quando esse executou os testes com o vetor ordenado. O *BoxPlot*(gráfico de caixa) mostrado na Figura 1 representa os dados da variavel média da Tabela 2. A variável média é a Média Aritmética dos cem testes realizados com cada tamanho de vetor. É possível observar que os menores tempos de execução são obtidos pela Busca Binária. E que a mediana que é uma medida (linha preta que fica em destaque) de tendência central da Estatística usada pelo BoxPlot mostra a Busca por Saltos e a Busca Binária tem o mesmo valor, ou seja em geral tiveram desempenhos de busca médio parecidos. Entretando a média do algoritmo de Busca Sequencial cresce exponencialmente. Assim, esse algoritmo pode ser até bom quando o tamanho do vetor é pequeno, mas quando o vetor é grande, a demora é inevitável [IME].

Referências

[IME] Algoritmos de busca em tabelas. https://www.ime.usp.br/~mms/mac1222s2013/15%20-%20algoritmos%20de%20busca%20em%20tabelas%20-%20sequencial%20e%20binaria.pdf. Accessed: 2021-04-05.

[Gee] Jump search. https://www.youtube.com/watch?v=63kS6ZkMpkA. Accessed: 2021-04-05.

[Git] Repositorio trabalho estrutura de dados - mestrado unioste. https://github.com/diogopaza/atividade_2_estrutura_de_dados_mestrado. Accessed: 2021-04-05.