Relatório Trabalho 1
Algoritmos e Estrutura de Dados II

Selton Miranda Rolim
GRR20235688

I. Introdução

O computador moderno foi planejado para armazenar e realizar diversas operações sobre os dados fornecidos como entrada. No entanto, para que seja utilizado de maneira eficiente é necessário que os dados estejam organizados em estruturas abstratas que permitam sua rápida leitura ou acesso. Portanto, a escolha de um algoritmo influência completamente a maneira que estruturamos estes dados.

A análise inicial começará pelos algoritmos de busca, visto que é uma tarefa frequente na computação e influenciará a escolha dos algoritmos de ordenação.

II. Método Experimental

Foram realizados ensaios nas mais diversas situações de entrada dos dados, ordenados, inversamente ordenados, aleatórios e quase ordenados, o último sendo 50% ordenado e 50% desordenado, permitindo maior generalização. A cada teste submetido, o volume de elementos era incrementado e foram registrados o tempo necessário para a execução e a quantidade de comparações realizadas. Para os algoritmos de busca, foi determinado que os vetores estão previamente ordenados e os elementos a serem buscados, aleatórios.

III. Busca Sequencial e Binária

Os dados obtidos dos experimentos realizados com a busca sequencial (BS) e binária (BB) para encontrar o mesmo elemento, a estão mostrados na tabela a seguir. Foram registrados o tempo, em segundos, e quantidade de comparações entre elementos do vetor.

Tamanho do vetor	10^6	10^8	10^9
Elemento	619736	77595141	747117287
BS Tempo (s)	0.0016	0.2000	2.00000
BB Tempo (s)	0.000001	0.000003	0.000003
BS Comparações	619737	77594142	747117288
BB Comparações	21	28	31

Tabela I
Diferenças Busca Sequencial e Busca Binária.

Os dados apresentados na Tabela I evidenciam que, à medida que o tamanho do vetor aumenta, o tempo necessário para realizar a busca permanece relativamente estável no contexto da busca binária, grande parte responsável ao custo da busca ser $\log_2 n + 1$ [1]. Em contrapartida, a busca sequencial demonstra uma tendência oposta, caracterizada pelo aumento significativo no tempo de busca à medida que o tamanho do conjunto de dados aumenta.

Esse fenômeno é atribuído ao fato de que a busca sequencial requer um número excessivo de comparações para localizar um elemento específico. Esses resultados corroboram a superioridade da busca binária sobre a busca sequencial, destacando a importância da eficiência do algoritmo na manipulação de grandes volumes de dados

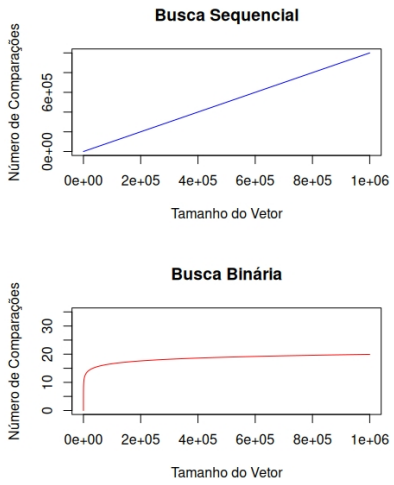

Figura 1. Relação Número de comparações e Tamanho do vetor

Os resultados dos testes conduzidos com a busca sequencial recursiva revelaram limitações severas, uma vez que a abordagem resultou em um aumento considerável na alocação de memória devido à recursividade. Este fator desencoraja a sua utilização prática, uma vez que a escalabilidade da técnica é comprometida pela carga adicional imposta ao sistema.

IV. Ordenação

Foram conduzidos ensaios em vetores ordenados, inversamente ordenados, aleatórios e quase ordenados. A partir disso, registraram-se os tempos de execução e o número de comparações para cada algoritmo.

A. *Insertion Sort*

Tamanho	10^5	10^6	10^8	10^9
Tempo (Segundos)	0.006	0.05	0.40	4.00
Comparações	100 mil	1 milhão	100 milhões	1 bilhão

Tabela II
Vetor Ordenado

Tamanho	10^5	2.10^5	3.10^5	4.10^5
Tempo (Segundos)	14.00	58.00	128.00	224.00
Comparações	5 bilhões	20 bilhões	45 bilhões	80 bilhões

Tabela III
VETOR INVERSAMENTE ORDENADO

Tamanho	10^5	2.10^5	3.10^5	4.10^5
Tempo (Segundos)	7.00	27.00	63.00	115.00
Comparações	2.5 bilhões	10 bilhões	22.5 bilhões	40 bilhões

Tabela IV
VETOR ALEATORIAMENTE DESORDENADO

Tamanho	10^5	2.10^5	3.10^5	4.10^5
Tempo (Segundos)	4.00	14.00	31.00	57.00
Comparações	1.25 bilhões	5 bilhões	11.2 bilhões	20 bilhões

Tabela V
VETOR QUASE ORDENADO

O Insertion Sort exibe um comportamento único, demonstrando caráter linear em cenários onde o vetor a ser ordenado já se encontra ordenado, *i.e.* o melhor caso. No entanto, quando o vetor está desordenado, o desempenho do algoritmo torna-se quadrático, *i.e.* o pior caso. Este fenômeno destaca a sensibilidade do Insertion Sort à disposição inicial dos dados. Em situações onde os dados estão próximos da ordenação, o Insertion Sort demonstra uma vantagem comparativa em relação a outros algoritmos de ordenação [2].

B. Selection Sort

Tamanho	10^5	2.10^5	3.10^5	4.10^5
Tempo (Segundos)	11.00	44.00	112.00	208.00
Comparações	5 bilhões	20 bilhões	45 bilhões	80 bilhões

Tabela VI
DADOS SELECTION SORT

O Selection Sort, embora apresente simplicidade de implementação e baixo consumo de recursos computacionais, exibe um comportamento de complexidade quadrática em relação ao número de elementos a serem ordenados. Esta característica implica que o número de comparações realizadas pelo algoritmo segue uma taxa de crescimento quadrática, expressa pela fórmula $n^2/2$, para qualquer caso, onde n representa o tamanho do vetor de entrada. Ademais, o tempo de execução mantém-se estável, com pouca variação, para qualquer tipo de entrada dos dados.

Essa complexidade quadrática torna o Selection Sort inadequado para lidar com conjuntos de dados extensos, uma vez que seu desempenho reduz rapidamente à medida que o tamanho do vetor aumenta. Sendo eficiente, no entanto, para conjuntos de dados reduzidos.

C. Merge Sort

Tamanho	10^6	10^7	10^8	10^9
Tempo	0.16	2.5	21.00	238.00
Comparações	20 milhões	233 milhões	2.67 bilhões	30 bilhões

Tabela VII
DADOS MERGE SORT

O algoritmo de ordenação Merge Sort demonstra um comportamento logarítmico em relação ao número de comparações, seguindo a complexidade teórica de $n * log_2 n$ em qualquer cenário. Apesar de sua eficiência em lidar com grandes volumes de dados, a principal desvantagem associada ao Merge Sort reside em seu consumo de memória adicional. Isso decorre da necessidade de um vetor auxiliar para a ordenação, resultando em um custo de memória aproximado de $2 * n$. Assim como o Selection Sort, o tempo de execução mantém-se estável qual seja a entrada de dados, com pouca variação.

Dada a natureza "Dividir e Conquistar"do Merge Sort, surge uma estratégia interessante quando o vetor de entrada atinge dimensões reduzidas. Nesse ponto, a utilização do algoritmo Insertion Sort torna-se vantajosa para a ordenação dos elementos, devido à sua complexidade linear, especialmente em cenários de melhor caso.

Essa abordagem híbrida, combinando o Merge Sort para grandes conjuntos de dados e o Insertion Sort para vetores de tamanho reduzido, visa otimizar o desempenho global do processo de ordenação, aproveitando as vantagens de cada algoritmo em diferentes escalas de problema [3].

V. CONCLUSÃO

O Merge Sort apresenta a melhor eficiencia dentre os algoritmos apresentendos, seu consumo de memória, no entanto, torna-o desvantajoso para cenários de poucos recursos. O Insertion Sort lida muito bem com cenário em que o vetor está praticamente ordenado. Por fim, o Selection Sort é o menos eficiente, tornando-o mais utilizável em vetores de tamanho reduzidos. Os equivalentes recursivos do Selection Sort e Insertion Sort não apresentaram dados suficientemente significativos, visto que ocorria falha de segmentação próximo do tamanho 200 mil. Além disso, não foi obter possível obter resultados satisfatórios para conjuntos superiores à 1 milhão, uma vez que estes últimos algoritmos, tornavam-se lentos demais, devido à sua complexidade quadrática.

No que diz respeito à busca binária, sua vantagem reside na sua complexidade logarítmica, o que a torna altamente eficiente em praticamente todos os cenários de busca em conjuntos de dados ordenados.

Assim, a escolha entre esses algoritmos depende das características específicas do problema em questão, considerando fatores como o tamanho do conjunto de dados, a ordem inicial dos elementos e as restrições de recursos computacionais disponíveis.

REFERÊNCIAS

[1] R. Sedgewick, *Algorithms in C*. Addison-Wesley Professional, January 1, 1990.

[2] ——, *Algorithms in C*, 1st ed. Addison-Wesley Professional, January 1, 1990.

[3] R. L. R. C. S. Thomas H. Cormen, Charles E. Leiserson, *Introduction to Algorithms*. The MIT Press, Setember 1, 2009.