RA4 - Ordenação

Marcelo Wzorek Filho

December 11, 2023

Abstract

Este trabalho tem como objetivo realizar uma análise de desempenho de algoritmos de ordenação. O estudo visa apresentar uma avaliação abrangente, considerando diversos aspectos em diferentes algoritmos. Os resultados obtidos visão fornecer insights valiosos que podem orientar a escolha do algoritmo de ordenação mais adequado para situações específicas, contribuindo assim para a melhoria do processamento de dados em sistemas computacionais.

1 Introdução

A ordenação de dados é uma operação fundamental em ciência da computação e desempenha um papel crucial em diversas aplicações, desde a organização de bancos de dados até a otimização de algoritmos de busca. A escolha do algoritmo de ordenação apropriado pode afetar significativamente o desempenho de um sistema ou aplicação. Neste contexto, este trabalho se concentra na análise de desempenho de três algoritmos de ordenação amplamente utilizados: o Insertion Sort, o Quick Sort e o Shell Sort.

Este estudo visa apresentar uma análise abrangente do desempenho desses algoritmos em termos de tempo de execução, número de trocas e número de iterações. Ao compreender as características e eficiência desses algoritmos, os profissionais de ciência da computação e engenheiros de software podem tomar decisões informadas ao selecionar o algoritmo de ordenação mais adequado para as necessidades específicas de suas aplicações.

O restante deste artigo está organizado da seguinte maneira: na seção 2, apresentaremos uma revisão da literatura relacionada aos algoritmos de ordenação discutidos; na seção 3, descreveremos a metodologia e os experimentos realizados; na seção 4, apresentaremos e discutiremos os resultados obtidos; e, por fim, na seção 5, concluiremos com uma síntese das descobertas e possíveis direções futuras de pesquisa. A implementação do algoritmo utilizado está disponível no meu Git Hub.

2 Conceito dos Algoritmos de Ordenação

2.1 Insertion Sort

O Insertion Sort é um algoritmo de ordenação simples e eficaz que opera comparando e movendo elementos um por um para a posição correta dentro de uma lista. Ele é particularmente eficiente em conjuntos de dados pequenos e quase classificados. No entanto, sua complexidade aumenta

consideravelmente à medida que o tamanho do conjunto de dados aumenta. Neste trabalho, examinaremos o desempenho do Insertion Sort em uma variedade de cenários e analisaremos o tempo de execução, o número de trocas e o número de iterações envolvidas neste algoritmo.

2.2 Quick Sort

O Quick Sort é amplamente reconhecido por sua velocidade e eficiência na ordenação de grandes conjuntos de dados. Utilizando a estratégia de divisão e conquista, o Quick Sort é capaz de classificar dados em tempo médio mais rápido do que muitos outros algoritmos de ordenação. Nesta pesquisa, investigaremos a eficácia do Quick Sort em diferentes situações, examinando o tempo de execução, o número de trocas e o número de iterações. A compreensão do desempenho do Quick Sort é fundamental para a seleção de algoritmos de ordenação em aplicações de alto desempenho.

2.3 Shell Sort

O Shell Sort é uma variação do Insertion Sort que melhora seu desempenho, especialmente em conjuntos de dados maiores. Ao dividir o conjunto de dados em subconjuntos menores e aplicar o Insertion Sort a esses subconjuntos, o Shell Sort consegue combinar eficiência e simplicidade. Neste estudo, investigaremos o desempenho do Shell Sort, considerando o tempo de execução, o número de trocas e o número de iterações em diferentes cenários. Compreender as características do Shell Sort é crucial para escolher o algoritmo de ordenação certo para diversas aplicações.

3 Metodologia e Experimentos

3.1 Descrição da Metodologia

Nesta seção, descreveremos a metodologia adotada para conduzir os experimentos de análise de desempenho dos algoritmos de ordenação. Para avaliar o desempenho do Insertion Sort, Quick Sort e Shell Sort, utilizamos uma abordagem sistemática. Primeiramente, preparamos conjuntos de dados de entrada contendo vetores de inteiros. Os vetores foram preenchidos aleatoriamente com 50, 500, 1000, 5000 e 10000 elementos, abrangendo diferentes tamanhos de entrada que são frequentemente encontrados em aplicações reais. Em seguida, implementamos os algoritmos de ordenação em um ambiente de desenvolvimento Java e utilizamos a aplicação de monitoramento VisualVM para coletar dados de desempenho, incluindo métricas como tempo de execução, consumo de memória e uso da CPU. Cada algoritmo foi executado em todos os tamanhos de conjunto de dados para obter resultados abrangentes. Para garantir a confiabilidade dos resultados, repetimos cada experimento várias vezes. A metodologia detalhada apresentada nesta seção serviu como base para a realização de experimentos confiáveis.

3.2 Configuração Experimental

A configuração experimental desempenha um papel crítico na avaliação do desempenho dos algoritmos de ordenação. Nossos experimentos foram realizados em um ambiente controlado e consistente para garantir resultados precisos e comparáveis. Utilizamos um computador Dell Inspiron i5 3520 com as seguintes especificações:

- Processador: 11th Gen Intel(R) Core(TM) i5-113
- Memória RAM: 16 GB
- Armazenamento: 512 GB
- Sistema Operacional: Ubuntu

Para coletar dados de desempenho em tempo real, empregamos a ferramenta VisualVM, que forneceu informações detalhadas sobre o consumo de recursos durante a execução dos algoritmos. Os conjuntos de dados de entrada foram criados aleatoriamente com números inteiros e variaram em tamanho de 50 a 10.000 elementos, como mencionado anteriormente.

4 Resultados e Discussão

Nossa análise de desempenho dos algoritmos de ordenação utiliza cinco conjuntos de dados de 50 a 10.000 elementos de valor inteiro e aleatório e trabalha com a media aritmética de dados coletados como CPU, queé medida em porcentagem (%), Tempo em microssegundos (μs), Trocas e Iterações.

Insertion Sort

Table 1: Desempenho do algoritmo Insertion Sort

	CPU (%)	Tempo (μs)	Trocas	Iterações
DADOS 50	1.1	60	645	49
DADOS 500	1.4	4274	62274	499
DADOS 1000	1.9	6958	241088	999
DADOS 5000	2.8	33955	6273138	4999
DADOS 10000	4.2	81563	25050714	9999

Quick Sort

Table 2: Desempenho do algoritmo Quick Sort

	CPU (%)	Tempo (μs)	Trocas	Iterações
DADOS 50	1.3	46	154	260
DADOS 500	1.5	831	2786	4616
DADOS 1000	1.7	1508	6898	11040
DADOS 5000	2.2	4244	47778	71470
DADOS 10000	2.7	6626	128189	179896

Shell Sort

Table 3: Desempenho do algoritmo Shell Sort

	CPU (%)	Tempo (μs)	Trocas	Iterações
DADOS 50	1.3	33	168	203
DADOS 500	1.4	586	2855	3506
DADOS 1000	1.7	1571	7202	8006
DADOS 5000	2.3	7274	54368	55005
DADOS 10000	3.1	10192	138099	120005

5 Conclusão e Considerações Finais

A análise de desempenho dos algoritmos de ordenação revela informações valiosas sobre o comportamento desses algoritmos em diferentes cenários.

Insertion Sort

O Insertion Sort é um algoritmo simples e eficiente para conjuntos de dados pequenos. Ele tem um baixo uso de CPU e é rápido para classificar pequenos conjuntos de dados. No entanto, seu desempenho degrada rapidamente à medida que o tamanho dos dados aumenta, como foi demonstrado acima. Para conjuntos de dados maiores, o Insertion Sort se torna menos prático devido ao alto número de trocas necessárias.

Quick Sort

O Quick Sort se destaca por sua eficiência e é uma excelente escolha para conjuntos de dados de tamanho moderado a grande. Ele é rápido e tem um uso razoável de CPU. No entanto, o Quick Sort não é estável e pode apresentar desempenho irregular em conjuntos de dados já quase ordenados, isso foi visível durante o período de teste, já que o uso de CPU saltava de maneira imprevisivel.

Shell Sort

O Shell Sort é um algoritmo intermediário que oferece um equilíbrio entre o Insertion Sort e o Quick Sort. Ele é mais eficiente do que o Insertion Sort para conjuntos de dados maiores e tem um uso moderado de CPU. O Shell Sort é uma escolha sólida quando o desempenho do Quick Sort pode ser um problema, como em conjuntos de dados quase ordenados.

Escolhendo o Algoritmo Certo

A escolha do algoritmo de ordenação apropriado depende do tamanho dos dados, da distribuição dos dados e das limitações de recursos, como a disponibilidade de CPU. Aqui estão algumas diretrizes gerais:

Para pequenos conjuntos de dados ou conjuntos de dados quase ordenados, o Insertion Sort pode ser uma escolha eficaz devido à sua simplicidade.

Para conjuntos de dados de tamanho moderado a grande, o Quick Sort é frequentemente a escolha preferida devido à sua eficiência geral.

Quando a estabilidade é necessária ou quando os dados estão quase ordenados, o Shell Sort pode ser uma alternativa adequada ao Quick Sort.

Em resumo, não há um único algoritmo de ordenação que seja o melhor em todas as situações. A escolha do algoritmo depende das características dos dados e dos requisitos de desempenho específicos do seu aplicativo.