Estrutura de Dados - Aula Prática 07 - Relatório

Lucas de Oliveira Araújo, 2022036039

<2023-05-22 Mon>

Contents

---

1 Introdução

A organização de dados é uma das partes mais importantes da computação. Isso se faz, pois a recuperação eficiente destes dados depende diretamente de alguma estrutura que os organiza de forma lógica, isto é, organizá-los conforme um conjunto específico de regras que permita determinar onde o tal dado está localizado, ou pelo menos onde ele tem mais probabilidade de estar.

Seguindo essa lógica, há diversos algoritmos de ordenação, alguns mais sofisticados e eficientes do que outros. Nesse trabalho, vamos realizar uma análise comparativa dos tempos de execução de dois desses algoritmos, a saber, o Shell Sort e o Heap Sort.

Shell Sort é um algoritmo de ordenação baseado no método de inserção direta do algoritmo Insertion Sort, porém com uma estratégia de dividir para conquistar. Essa estratégia previne que a ordenação tenha a complexidade $O(n^2)$, que é o pior caso do Insertion Sort. Contudo, devido à dinâmica de como essa estratégia pode ser implementada, a complexidade do Shell Sort não é conhecida para todos os casos.

O Heap Sort, por outro lado, tem complexidade $O(n \log n)$ para todos os casos. Sendo assim, ele se mostra um bom parâmetro de comparação de tempo de execução para com outros algoritmos. Dessa maneira, vamos analisar o desempenho do Shell Sort para uma determinada estratégia de divisão e conquista usando o Heap Sort como régua para nossa análise.

2 Plano de testes

O método de divisão e conquista do Shell Sort é feito determinando tamanhos de intervalos de ordenação diferentes, isto é, distâncias entre os números de um certo array. Nesse sentido, o intuito é que os intervalos diminuam até chegar no intervalo de tamanho 1, momento no qual o Shell Sort se comportará como o Insertion Sort. Entretanto, quando o intervalo for de tamanho 1, o array certamente estará ao menos parcialmente ordenado e, portanto, o Insertion Sort nunca se apresentará no seu pior caso.

Dito isso, o método de atualização dos intervalos é conhecido como H. Para nossa análise, escolhemos o método de atualização que utiliza-se do Número de Ouro, também conhecido como *Golden Ratio* ou Proporção Áurea, descrito matematicamente pela grega Phi e dado pela fórmula $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803\ldots$. Tal número é amplamente notável pela simetria na qual objetos e composições que seguem essa proporção adquirem.

Dessa maneira, nosso H é criado formando potências de ϕ de forma que chegue na potência mais próxima mas menor do que o tamanho do array. Por se tratar de potências e por ser um número inicialmente pequeno, o escolhemos pois, para os primeiros expoentes, o tamanho de H, isto é, as distâncias, são também pequenas. Por outro lado, para expoentes a partir de 10, os intervalos aumentam significativamente. Dessa maneira, fazendo o caminho inverso, ou seja, pegar a maior potência menor do que o tamanho do array, poderemos ordenar os números que estão mais distantes das posições em que eles deveriam estar e, à medida com que H tenda à 1, os intervalos vão diminuindo de forma que o array seja incrementalmente ordenado até o estar por completo.

Definido o método de atualização do H, a implementação propriamente dita foi feita utilizando-se de duas classes:

1. *SortUtils*: Classe que contém os métodos de ordenação Shell Sort com a estratégia de divisão e conquista citada;
2. *Analyzer*: Classe que contém métodos utilizados para analisar os métodos de ordenação conforme um certo tamanho de array.

Mais especificamente, a classe Analyzer contém todo o ferramental para a análise dos métodos de ordenação, desde o método de geração de um array desordenado de tamanho N até o método de plotagem dos gráficos com os tempos médios de execução dos algoritmos de ordenação a partir de uma bateria de testes.

3 Análise dos resultados

Os testes foram divididos em duas etapas:

1. Análise dos dois algoritmos para arrays de tamanhos N, tal que $1000 \leq N \leq 1000000$, de forma que N varia de 10 em 10 mil. Para cada array de tamanho N, foram executadas uma bateria de 10 testes, no intuito de diluir possíveis outliers, isto é, a geração de um array mais facilmente de ordenar para um algoritmo e um outro array discrepantemente mais difícil de ordenar para o outro algoritmo.
2. Análise dos dois algoritmos para arrays de tamanhos N, tal que $1000000 \leq N \leq 25000000$, de forma que N varia de 1 em 1 milhão. Da mesma maneira que a primeira etapa de testes, para cada array de tamanho N, foram executadas uma bateria de 10 testes.

O intuito da primeira etapa é verificar o desempenho dos dois algoritmos para arrays relativamente pequenos. Por outro lado, o segundo teste visa verificar o desempenho de tais algoritmos para arrays grandes e mais custosos de ordenar.

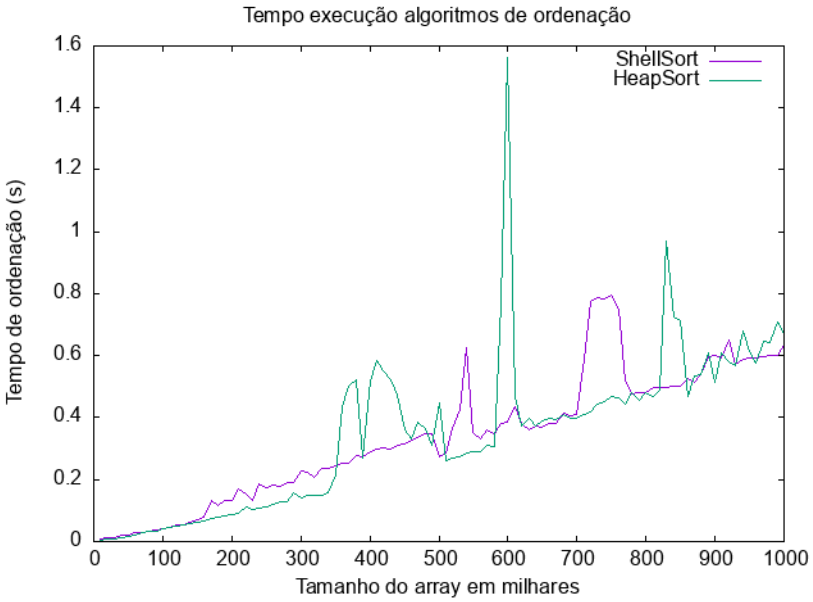

Figure 1: Tempos médios de execução da primeira etapa de testes

Na figura 1, a qual representa os resultados da primeira etapa de teste, é possível notar bastante instabilidade nos tempos de execução dos dois algoritmos conforme o N aumenta. Apesar disso, os dois algoritmos seguem a mesma tendência, o que pode ser usado para considerar um desempenho equiparável entre ambos.

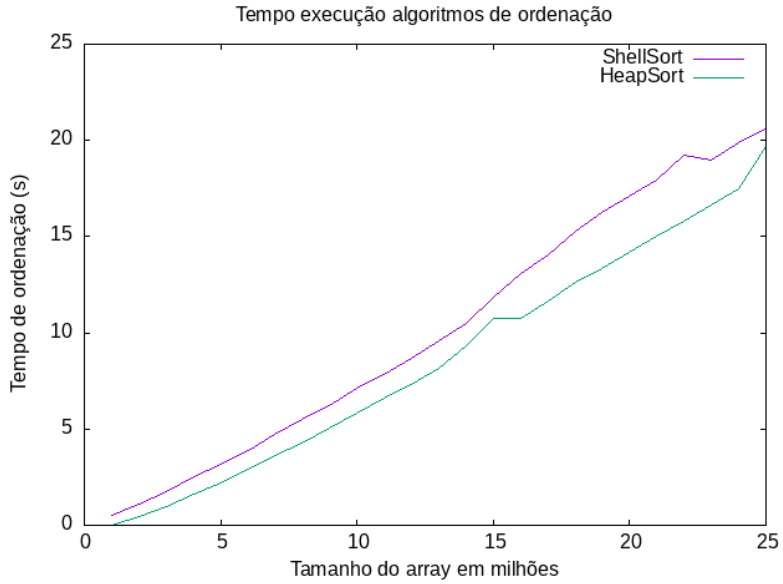

Figure 2: Tempos médios de execução da segunda etapa de testes

Por outro lado, a segunda etapa de teste mostrou que o Heap Sort foi superior ao Shell Sort durante todo o intervalo de arrays com N na casa dos milhões. Apesar disso, o Shell Sort se apresenta com

a mesma tendência de subida que o Heap Sort. Portanto, apesar de ser mais custoso do que o Heap Sort, o Shell Sort ainda se mostra competitivo conforme o N aumenta.

Em suma, a estratégia de atualização do H escolhida para o Shell Sort o tornou equiparável ao Heap Sort para arrays menores que 1 milhão, mas para arrays maiores o Heap Sort ainda se mantém mais eficiente.

4 Conclusão

Este trabalho apresentou os algoritmos de ordenação Heap Sort e Shell Sort, além de realizar uma análise comparativa entre os tempos médios de execução dos dois algoritmos para arrays de diferentes tamanhos.

Pela análise realizada, conclui-se que para arrays consideravelmente pequenos ($N \leq 1000000$) o método de atualização do Shell Sort que utiliza-se o Número de Ouro se mostra uma boa escolha. Em contraste, para arrays com milhões de elementos essa estratégia de atualização não se mostra tão eficiente.

5 Referências

1. WIKIPEDIA CONTRIBUTORS. Shellsort. Disponível em: `https://en.wikipedia.org/wiki/Shellsort`. Acesso em: 21 maio 2023.

2. Correção e desempenho do algoritmo Heapsort. Disponível em: `https://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html`. Acesso em: 22 maio 2023.