ALGORITMO DE ORDENAÇÃO COMB SORT

Notação Big O

Conceito matemático amplamente utilizado para avaliar a eficiência de um algoritmo. É uma ferramenta simples e poderosa para classificar a complexidade computacional de um algoritmo.

Na área da ciência da computação, o Big O é aplicado para categorizar algoritmos com base no seu tempo de execução (número de passos) ou nos requisitos de espaço, dependendo das variações nos dados de entrada. Essa classificação permite uma análise objetiva e uma comparação entre diferentes algoritmos, facilitando a escolha da melhor abordagem para resolver problemas computacionais específicos. É importante considerar o Big O ao projetar algoritmos eficientes, pois ele fornece insights valiosos sobre o desempenho e a escalabilidade dos mesmos.

Por que é importante?

O Big O desempenha um papel fundamental no desenvolvimento de software, permitindo melhorar a qualidade dos programas ao compreendermos como é possível otimizar o desempenho. Isso torna o software mais eficiente, escalável e livre de complexidades desnecessárias.

Ao lidar com um projeto que requer a implementação de um algoritmo específico, é essencial compreender o quão rápido ou lento ele é em comparação com outras alternativas disponíveis. Essa compreensão nos ajuda a tomar decisões informadas sobre a escolha do algoritmo mais adequado, levando em consideração a eficiência e a capacidade de escalabilidade.

Funções e Complexidade Computacional

Na análise da complexidade de algoritmos, a variável "n" representa o tamanho da entrada, enquanto a função associada ao "O()" indica a complexidade do algoritmo em relação a esse tamanho.

Por exemplo, quando dizemos que um algoritmo possui complexidade O(n), significa que o tempo de acesso a um elemento é proporcional à quantidade de elementos na coleção.

É importante ressaltar que o Big O **não** mede a velocidade de um algoritmo em segundos, pois existem diversos fatores, como o hardware utilizado para executar o algoritmo, que podem influenciar essa velocidade.

Valores de desempenho para vetores

A código implementado em Linguagem C, foi rodado com 10 vetores em ordem aleatória, em ordem crescente e ordem decrescente. Os resultados obtidos foram transcritos para a tabela a abaixo. Da mesma forma foi feito a simulação com vetores de tamanho 1.000 e 10.000 nas respectivas ordens (aleatória, crescente e decrescente). Chegando assim a uma classificação de casos a seguir comentadas.

	Ordem Aleatória		Ordem Crescente		Ordem Decrescente	
Tam. Vetor	Comparações	Trocas	Comparações	Trocas	Comparações	Trocas
10	41	11	32	0	41	7
1.000	22.709	4.178	18.713	0	19.712	1.582
10.000	336.733	60.148	276.739	0	286.738	20.078
	Caso Médio		Melhor Caso		Pior Caso	

Caso	Notação Big O	Descrição
Caso Médio	O (n^2)	Tempo de execução cresce **quadraticamente** com o tamanho da entrada.
Melhor Caso	O (n log n)	Tempo de execução cresce de forma **logarítmica** com o tamanho da entrada.
Pior Caso	O (n^2)	Tempo de execução cresce **quadraticamente** com o tamanho da entrada.

Conforme evidenciado na tabela, o pior caso para o algoritmo Comb Sort ocorre quando o vetor está ordenado em ordem decrescente. Nessa situação, o algoritmo requer um número máximo de comparações e trocas. Isso acontece porque, a cada iteração, o maior elemento é movido apenas uma posição por vez em direção ao final da lista. Portanto, o pior caso do Comb Sort tem uma complexidade de tempo de O(n^2), onde "n" representa o tamanho do vetor. Essa notação quadrática indica um desempenho mais lento à medida que o tamanho do vetor aumenta.

Por outro lado, o melhor caso ocorre quando o vetor já está ordenado em ordem crescente. Nesse cenário, o algoritmo não precisa realizar trocas, pois

cada elemento já está no lugar correto. Assim, o melhor caso do Comb Sort tem uma complexidade de tempo de O(n), indicando um tempo de execução linear e proporcional ao tamanho do vetor. Isso resulta em um desempenho muito mais eficiente em comparação com o pior caso.

No caso médio, que representa a ordenação aleatória mais comum, o desempenho do Comb Sort geralmente fica entre o melhor e o pior caso. Embora não seja possível atribuir uma notação Big O precisa, ao caso médio, podemos considerar que o Comb Sort possui uma complexidade média de tempo entre O(n) e O(n^2), dependendo da distribuição dos elementos no vetor. Em geral, o Comb Sort tende a ter um desempenho melhor do que outros algoritmos de ordenação quadráticos, mas não é tão eficiente quanto algoritmos de ordenação mais avançados, que possuem complexidade média de tempo O (n log n).

INTRODUÇÃO AO ALGORITMO COMB SORT

O algoritmo Comb Sort, também conhecido como algoritmo do pente, é um método de ordenação simples pertencente à categoria dos algoritmos de troca. Foi desenvolvido por Wlodzimierz Dobosiewicz em 1980 e posteriormente popularizado por Stephen Lacey e Richard Box através de um artigo na revista Byte em abril de 1991. O Comb Sort é uma evolução do Bubble Sort e rivaliza diretamente com o Quick Sort.

Método Comb Sort

O Comb Sort é um algoritmo de ordenação que utiliza o método de reordenar pares de dados separados por um intervalo, o qual é calculado a cada passagem. Essa técnica torna o Comb Sort similar ao Bubble Sort, porém, mais eficiente. Enquanto no Bubble Sort os dados são comparados sempre com um intervalo fixo de 1, no Comb Sort esse intervalo pode ser maior, permitindo uma reorganização mais rápida dos elementos. O cálculo do intervalo no Comb Sort é determinado por um fator de redução, que contribui para a otimização do algoritmo. Com a redução do número de trocas necessárias, o Comb Sort apresenta um desempenho aprimorado em relação ao Bubble Sort.

Coelhos e Tartarugas

Os elementos com valores grandes próximos ao início da lista são chamados de coelhos, enquanto os elementos com valores pequenos próximos ao final são chamados de tartarugas. A ideia principal é eliminar as tartarugas, ou seja, os valores pequenos próximos ao final da lista, pois eles podem prejudicar o desempenho da ordenação. Os coelhos, que são os valores grandes próximos ao início, não afetam negativamente o processo de ordenação.

Iteração	Vetor
0	[9, 3, 4, 0, 8, 7, 6, 5, 1, 2]
1	[3, 4, 0, 8, 7, 6, 5, 1, 2, 9]
2	[3, 0, 4, 7, 6, 5, 1, 2, 8, 9]
3	[0, 3, 4, 6, 5, 1, 2, 7, 8, 9]
4	[0, 3, 4, 5, 1, 2, 6, 7, 8, 9]
5	[0, 3, 4, 1, 2, 5, 6, 7, 8, 9]
6	[0, 3, 1, 2, 4, 5, 6, 7, 8, 9]
7	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

O elemento 9 é um coelho e o elemento 1 é uma tartaruga. Elementos mais pesados que devem ser movidos no final do vetor se movem muito rapidamente, enquanto elementos mais leves que devem ser movidos para o início do vetor se movem apenas uma etapa em cada iteração.

Fator de encolhimento

No artigo original, os autores propuseram o valor de 1,3 após testar diversas listas aleatórias (estima-se que tenham sido mais de 200.000 listas). Caso o valor desse fator seja muito baixo, o algoritmo se torna mais lento devido ao aumento no número de comparações necessárias. Por outro lado, se o valor do fator for muito alto, ele pode não lidar adequadamente com um número suficiente de valores menores, tornando-se impraticável em termos de desempenho. Uma medida cuidadosamente selecionada com base em extensos testes em listas aleatórias é o fator de encolhimento de 1,24 utilizado no algoritmo. Essa escolha estratégica visa encontrar um equilíbrio no desempenho do algoritmo, evitando tanto a lentidão excessiva quanto a ineficiência ao lidar com diferentes tamanhos de dados. Dessa forma, o fator de encolhimento de

1,24 desempenha um papel crucial na otimização do algoritmo de ordenação, garantindo sua eficácia e eficiência em diversas situações.

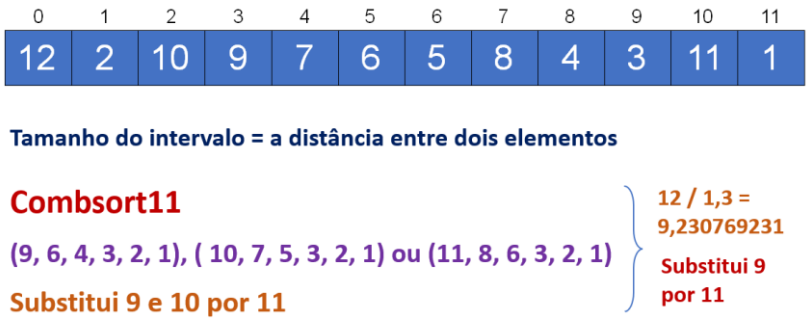
Como funciona...

O algoritmo Comb Sort é baseado no uso de uma variável chamada intervalo ou gap. Essa variável é inicializada com o comprimento do vetor dividido pelo fator de encolhimento, geralmente 1,3 ou 1,24. Os valores no vetor são ordenados considerando esse intervalo, que é arredondado para um número inteiro.

Esse processo de ordenação é repetido várias vezes, atualizando o intervalo a cada iteração, até que a diferença entre os elementos seja igual a 1. Nesse ponto, o Comb Sort continua a ordenação usando um intervalo fixo de 1, semelhante ao Bubble Sort. (Que compara 1 elemento à sua frente). Essa estratégia garante um equilíbrio entre eficiência e velocidade na ordenação dos dados.

Variações: Combsort11

Com um fator de encolhimento de aproximadamente 1,3, existem três possíveis sequências finais para a lista de intervalos: (9, 6, 4, 3, 2, 1), (10, 7, 5, 3, 2, 1) ou (11, 8, 6, 4, 3, 2, 1). Experimentos mostraram que é possível obter melhorias significativas de velocidade se o intervalo for definido como 11 sempre que for originalmente 9 ou 10. Essa variação é conhecida como Combsort11.



Se uma das sequências que começam com 9 ou 10 for utilizada, o passo final com intervalo 1 tem menor probabilidade de ordenar completamente os dados, exigindo assim outra iteração com intervalo 1. Os dados são considerados ordenados quando nenhuma troca adicional ocorrer durante uma passagem com intervalo igual a 1.