Um estudo sobre ordenadores

Maxwel Araujo Costa

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Câmpus Boituva, Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

maxwel_ara@yahoo.com

Resumo: Esse texto tem como objetivo analisar e comparar os principais ordenadores vistos em sala, na matéria de Estrutura de Dados II.

1. **Bubble Sort**

É um dos ordenadores mais simples e estáveis. Percorre o vetor analisando seus elementos por pares, e compara qual é maior. No caso de uma organização crescente, o Bubble Sort sempre transferirá o menor elemento do par para uma posição anterior ao maior, se o menor já não se encontrar em tal posição. “Após a primeira passagem completa pelo vetor (…) podemos garantir que o maior item terá sido deslocado para a última posição do vetor” [Pereira 2010, p.95]. Ele se aplica melhor nos casos de pequenos vetores, com poucos elementos a se comparar. Sua eficiência é $O(n^2)$.

2. **Insertion Sort**

O Insertion Sort é de fácil implementação, similar ao Bubble Sort (SZWARCFITER e MARKENZON, 2015). O Insertion Sort é um ordenador que percorre o vetor em diversas iterações, procurando um elemento que esteja fora de ordem, se encontrá-lo realiza a organização, repetindo o processo até analisar todos elementos do vetor. Ele se aplica melhor nos casos em que há poucos elementos a se comparar. Sua eficiência é $O(n^2)$.

3. **Selection Sort**

Um dos mais simples e utilizados (Jackson, João, Náthalee, 2017). O Selection Sort procura o menor item do vetor e o transfere para a primeira posição, depois procura o segundo menor item e o transfere para a segunda posição, e repete o processo até que o vetor esteja organizado. Ele se aplica melhor nos casos em que o vetor é pequeno. Sua eficiência é $O(n^2)$.

4. **Merge Sort**

Este algoritmo tem como objetivo a reordenação de uma estrutura linear por meio da quebra, intercalação e união dos n elementos existentes (Jackson, João, Náthalee, 2017). O Merge Sort divide o vetor em metades, sucessivamente até sobrar apenas vários grupos de pares, nesse ponto ele ordena os elementos dos pares e vai reconstruindo o vetor enquanto ordena os grupos de metades. Ele se aplica melhor nos casos em que grande quantidade de elementos a se comparar. Sua eficiência é $O (n \log n)$.

5. **Quick Sort**

O Quick Sort usa do mesmo princípio de divisão que o Merge Sort, entretanto, o mesmo não utiliza a intercalação, uma vez que não subdivide a dada estrutura em muitas menores (Jackson, João, Náthalee, 2017). O Quick Sort trabalha de maneira semelhante ao Merge Sort. Porém antes de realizar a divisão, ele seleciona um elemento para ser o pivô, este será incluído como centro do novo vetor e não participará da divisão, e a cada subdivisão um novo pivô é escolhido e posicionado próximo ao pivô original de acordo com sua posição espacial. Após todos os pivôs forem alocados, os números restantes são posicionados em ordem de resto de divisões mais ao centro, e mais recentes ao canto. Ele se aplica melhor nos casos em que grande quantidade de elementos a se comparar. Sua eficiência no pior caso é $O(n^2)$.

5. **Shell Sort**

Baseado, de forma melhorada, no Insertion Sort, o Shell Sort é um algoritmo que, fazendo uso de um valor de distanciamento, denominado gap, rearranja os elementos de uma estrutura por meio de inserção direta (SILVA, 2010). O Shell Sort disponibiliza uma opção de distância de salto, ela causa um impacto significativa no desempenho do ordenador, e a cada "salto" o ordenador realiza um Insertion Sort, até o vetor estiver ordenado. Ele se aplica melhor nos casos de arquivos de tamanho moderado. Sua eficiência consiste de O(n(log n) 2) ou O(n 1,25).

5. **Heap Sort**

O Heap Sort começa ao entender o vetor como se fosse uma árvore binária, ao separar o primeiro elemento e tratá-lo como raiz, os dois próximos como seus filhos, e assim sucessivamente. O Heap Sort possui duas modalidades: o Max-heap e o Min-heap. No Max-heap, o elemento raíz não pode ser menor do que seu filho da esquerda, por tanto se for o caso os elementos mencionados trocam de lugar; No Min- heap, exatamente o oposto acontece. Ele se aplica melhor nos casos em que há necessidade de controlar filas de prioridades. Sua eficiência consiste de O(n log n).

"Os filhos da raiz continuam sendo heaps máximos, mas o novo elemento raiz pode violar a propriedade de heap máximo. Porém, tudo o que é necessário para restabelecer a propriedade de heap máximo ´e uma chamada MaxHeapify(A,1), que deixa um heap máximo em A[1,... , n − 1]. Então, o algoritmo HeapSort repete esse processo para o heap de tamanho n − 1, descendo até um heap de tamanho 2" (ANDRÉ, 2008).

Bibliografia

Szwarcfiter, J. L. and Markezon, L. (2015). “Estruturas de Dados e Seus Algoritmos.” 3ª edição. Rio de Janeiro. LTC.

Pereira, S. L. (2010). “Algoritmos e Lógica de Programação em C: uma abordagem didática.” 1ª edição. São Paulo. Érica.

Silva, E. S. (2010). “Estudo Comparativo de Algoritmos de Ordenação.” 33 f. Trabalho de Conclusão de Curso – Universidade Federal do Espírito Santo, São Mateus.

Jackson É. G. Souza, João V. G. Ricarte, Náthalee C. A. Lima (2017). “Algoritmos de Ordenação: Um Estudo Comparativo” Universidade Federal Rural do Semi-Árido.

André Luis Trevisan (2008). “Algoritmos de Ordenação na Otimização do Valor Ordenado” Universidade Estadual de Campinas.