MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
CAMPUS HELVÍDIO NUNES DE BARROS – Picos
BACHARELADO DE SISTEMAS DE INFORMAÇÃO
DISCIPLINA: PROGRAMAÇÃO E ANÁLISE DE ALGORITMOS
DOCENTE: ISMAEL DE HOLANDA LEAL

Discentes: Héricly Sousa Borges; Luis Guilherme Sousa e Silva; Renan de Sousa Rêgo

COMPARAÇÃO DE DESEMPENHO NOS ALGORITMOS DE ORDENAÇÃO BUBBLE SORT, BUCKET SORT E COUNTING SORT

1 INTRODUÇÃO

Problemas são questões propostas em busca de uma solução. Com o propósito de conceder uma solução para certo problema, existem os algoritmos, cada problema que é decidível possui um algoritmo que determina uma solução para cada instância desse problema.

Um algoritmo é uma sequência de passos utilizados para resolver determinados problemas. Em certas situações um problema que já possui um algoritmo como solução, precisa ser resolvido com uma velocidade de processamento maior que a atual, ou seja, precisaremos de um outro algoritmo para resolver o mesmo problema, mas de forma diferente.

Um tipo de algoritmo muito usado na resolução de problemas computacionais são os algoritmos de ordenação, que servem para ordenar/organizar uma lista de números ou palavras de acordo com a sua necessidade. As linguagens de programação já possuem métodos de ordenação, mas é bom saber como funcionam os algoritmos, pois há casos de problemas em que o algoritmo de ordenação genérico não resolve, às vezes é necessário modificá-lo.

Neste artigo serão abordados três algoritmos de ordenação, que são eles: Bubble Sort, Bucket Sort, Counting Sort. Apontando suas definições, os aspectos históricos, a ordem de complexidade, comparação de funcionamento dos algoritmos e por fim os resultados obtidos na execução destes algoritmos numa mesma plataforma.

Este artigo está estruturado da seguinte maneira: na seção 2, estão sendo definidos cada um dos algoritmos de ordenação com seus respectivos aspectos históricos. Na Seção 3 é apresentada a ordem de complexidade de cada algoritmo. Na seção 4 é apresentado a comparação de funcionamento dos algoritmos. A seção 5 apresenta os resultados obtidos na execução dos mesmos. Encerramos nosso trabalho na seção 6, onde apresentamos a conclusão obtida diante dos resultados. E por fim, na seção 7 está nossa referência bibliográfica.

2 DEFINIÇÃO DOS ALGORITMOS

Os algoritmos de ordenação servem para colocar um conjunto de itens (geralmente numericamente ou lexicalmente) em uma ordem. A entrada é uma lista de itens e o retorno do algoritmo é a lista, de forma que deve seguir a ordenação estabelecida.

Vale lembrar também que a aplicação de algoritmos de ordenação são extremamente importantes para operações que incluem outros tipos de algoritmo tais como: busca binária e união de itens. Em outras palavras—existem algoritmos que possuem a premissa de que a lista esteja ordenada para assim aplicá-los.

2.1 Bubble Sort

O ***bubble sort***, ou ordenação por flutuação (literalmente "por bolha"), é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vector diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

Este algoritmo é um dos mais simples levando em conta a implementação, apesar de não ter um bom desempenho se comparado a outros tipos de algoritmos de ordenação. A ideia é sempre iterar toda a lista de itens quantas vezes forem necessário até que os itens estejam na ordem correta.

Este algoritmo percorre a lista de itens ordenáveis do início ao fim, verificando a ordem dos elementos dois a dois, e trocando-os de lugar se necessário. Percorre-se a lista até que nenhum elemento tenha sido trocado de lugar na passagem anterior, ele não consegue ordenar todo o vetor em uma única rodada, ele terá que passar pelo vetor um certo número de vezes:

1. Percorre o vetor inteiro comparando elementos adjacentes (dois a dois);
2. Troca as posições dos elementos se eles estiverem fora de ordem;
3. Repite os dois passos acima (n - 1) vezes, onde n é igual ao tamanho do vetor.

Ele não marca os valores já ordenados, ou seja ele percorre o vetor completamente em cada iteração, o que pode ser considerado um desperdício.

2.2 Bucket Sort

Bucket sort, ou **bin sort**, é um algoritmo de ordenação que funciona dividindo um vetor em um número finito de recipientes. Cada recipiente é então ordenado individualmente, seja usando um algoritmo de ordenação diferente, ou usando o algoritmo bucket sort recursivamente.

Bucket sort funciona do seguinte modo:

1. Inicialize um vetor de "baldes", inicialmente vazios.
2. Vá para o vetor original, incluindo cada elemento em um balde.
3. Ordene todos os baldes não vazios.

4. Coloque os elementos dos baldes que não estão vazios no vetor original.

A classificação de bucket é principalmente útil quando a entrada é distribuída uniformemente em um intervalo. Por exemplo, considere o seguinte problema. Classifique um grande conjunto de números de ponto flutuante que estejam no intervalo de 0,0 a 1,0 e sejam distribuídos uniformemente ao longo do intervalo.Uma maneira simples é aplicar um algoritmo de classificação baseado em comparação. O limite inferior para o algoritmo de classificação baseado em comparação (Merge Sort, Heap Sort, Quick-Sort, etc.) é Ω (n Log n), ou seja, eles não podem ser melhores que *n Log n*.

O Bucket Sort não é necessariamente/exatamente um algoritmo de ordenação, mas sim uma distribuição para tornar a organização mais efetiva. Por esse motivo é encontrado no código a função bubble, que é utilizada para organizar os elementos dentro de cada balde. Pode-se substituir o bubble sort por qualquer outro algoritmo de ordenação.

2.3 Counting Sort

Counting Sort é um algoritmo de ordenação estável. As chaves podem tomar valores entre 0 e M-1. Se existirem k0 chaves com valor 0, então ocupam as primeiras k0 posições do vetor final: de 0 a k0-1. Tem como uma de suas características a necessidade de se ter memória auxiliar. Logo, não é in-place.

O Counting Sort é uma técnica de classificação baseada em chaves entre um intervalo específico. Ele funciona contando o número de objetos com valores-chave distintos (tipo de hashing). Então fazendo alguma aritmética para calcular a posição de cada objeto na seqüência de saída. Pontos a serem observados:

1. A classificação de contagem é eficiente se o intervalo de dados de entrada não for significativamente maior que o número de objetos a serem classificados;
2. Não é uma classificação baseada em comparação. A complexidade do tempo de execução é O (n) com espaço proporcional ao intervalo de dados;
3. É freqüentemente usado como uma sub-rotina para outro algoritmo de classificação, como radix sort.
4. A contagem de contagem usa um hashing parcial para contar a ocorrência do objeto de dados em O (1).
5. O tipo de contagem também pode ser estendido para o trabalho de entradas negativas.

O poder do algoritmo está no fato de utilizar os próprios valores do vetor como índice em um outro vetor, desta forma que os valores ficam ordenados, porém a forma mais eficiente deste tipo de algoritmo se dá apenas com valores do tipo inteiro e que devem estar uniformemente espalhados no vetor de entrada para poder ser usado como índice.

3 ORDEM DE COMPLEXIDADE

3.1 Bubble Sort

No melhor caso, o algoritmo executa ***n*** operações relevantes, onde ***n*** representa o número de elementos do vetor. No pior caso, são feitas n^2 operações. A complexidade desse algoritmo é de ordem quadrática. Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

3.2 Bucket Sort

O Bucket Sort tem complexidade linear $\Theta(n)$ quando o vetor a ser ordenado contém valores que são uniformemente distribuídos.

3.3 Counting Sort

Complexidade: O(n)

4 COMPARAÇÃO DO FUNCIONAMENTO DOS ALGORITMOS

4.1 Bubble Sort

O algoritmo inicia comparando a primeira posição do vetor com a segunda posição do vetor. Se o elemento da segunda posição for menor que o da primeira, trocam-se as posições. Caso o elemento não seja, nada a se fazer.

Na próxima iteração, compara-se a segunda posição do vetor com a terceira posição do vetor. Se o elemento da terceira posição for menor que o da segunda, trocam-se as posições. Caso o elemento não seja, nada a se fazer.

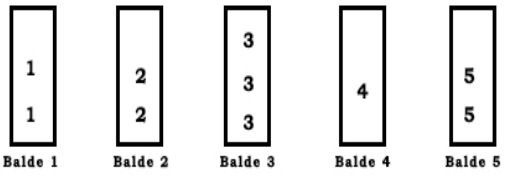
Repete-se essas iterações até o fim do vetor. Dessa forma é feita a primeira iteração por completo. Com podemos verificar, o elemento com maior valor ficou na última posição do vetor, esse processo vai se repetir até que o vetor esteja ordenado. Na próxima iteração, o segundo maior valor será ordenado para penúltima posição do vetor. Essa ordenação lembra como as bolhas num tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo, Bubble Sort (literalmente "flutuação por Bolha").

4.2 Bucket Sort

Um exemplo de ordenação por distribuição é o algoritmo bucket sort (ordenação balde). Esse algoritmo cria “baldes” com valores atribuídos a estes, esses baldes receberão os números referente ao seu valor depois estes são “esvaziados” seguindo a ordem.

Exemplo vetor = {2,1,3,5,1,3,3,5,4,2}

Levando em consideração o exemplo acima, seria necessário a criação dos baldes 1, 2, 3, 4 e 5. Os números seriam distribuídos nos baldes da seguinte forma:



Após isso, basta esvaziar os baldes a partir do primeiro para que os números fiquem na ordem.

4.3 Counting Sort

A ideia básica do counting sort é determinar, para cada entrada x, o número de elementos menor que x. Essa informação pode ser usada para colocar o elemento x diretamente em sua posição no array de saída. Por exemplo, se há 17 elementos menor que x, então x pertence a posição 18. Esse esquema deve ser ligeiramente modificado quando houver vários elementos com o mesmo valor, uma vez que nós não queremos colocar eles na mesma posição.

5. Testes

Para testes com os algoritmos de ordenação Bubble Sort, Bucket Sort e Counting Sort foi utilizado a linguagem C. Os algoritmos foram testados quatro vezes a cada 3 tamanhos de vetores (50k, 100k e 500k) para a obtenção da média em segundos. Os vetores foram preenchidos das seguintes formas:

- 1º Vetor: Foi utilizado como o melhor caso, ele foi preenchido já ordenado.
- 2º Vetor: Foi utilizado como o caso médio, ele foi preenchido de modo aleatório.
- 3º Vetor: Foi utilizado como o pior caso, ele foi preenchido com a ordem inversa.

Para a realização dos testes foi utilizado um computador com a seguinte configuração:

- **Processador:**
  - *Modelo:* Intel Core i5-5200U
  - *Memória cache:* 3MB
  - *Frequência base:* 2.20 GHz
- **Memória RAM:**
  - *Arquitetura:* DDR3 L
  - *Tamanho:* 8GB
- **Sistema Operacional:**
  - *Nome: Windows* 10 *Home Basic*
- **Softwares utilizados:**
  - *IDE:* Falcon C++
  - Versão 3.3

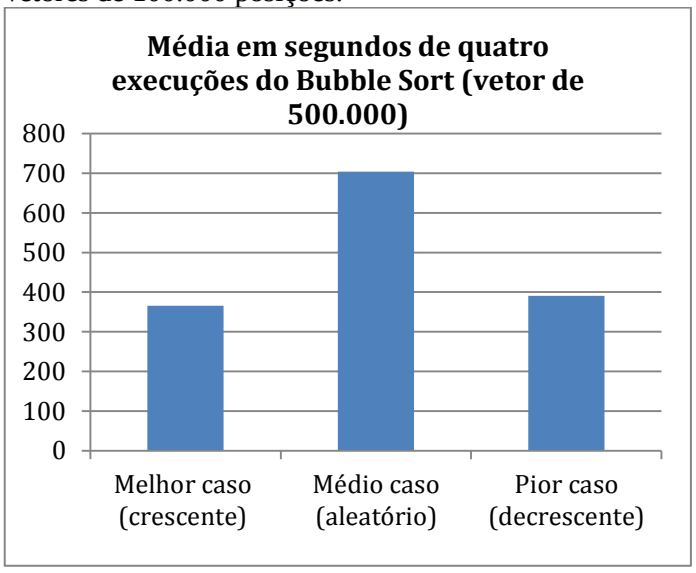
Nas tabelas a seguir podemos visualizar a média da saída de quatro execuções de cada algoritmo.

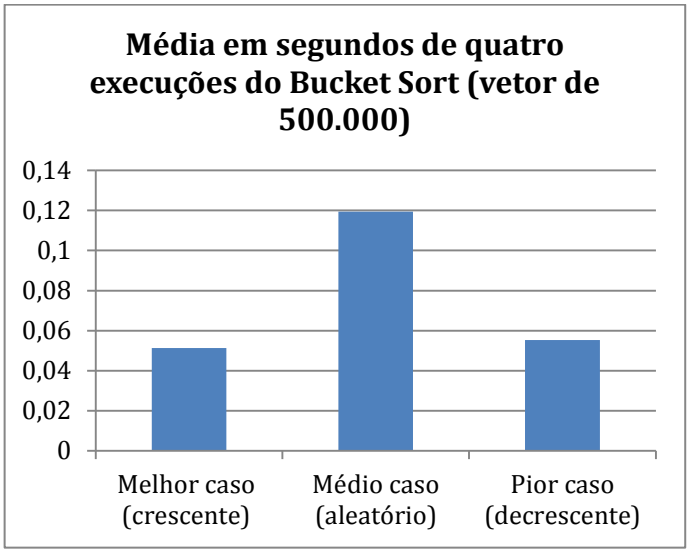
BUBBLE SORT			
VETOR	**Melhor caso (crescente)**	**Médio caso (aleatório)**	**Pior caso (decresc)**
50.000	5,1	8,125	4,4
100.000	16,325	30,275	17,325
500.000	365,25	704,325	390,375

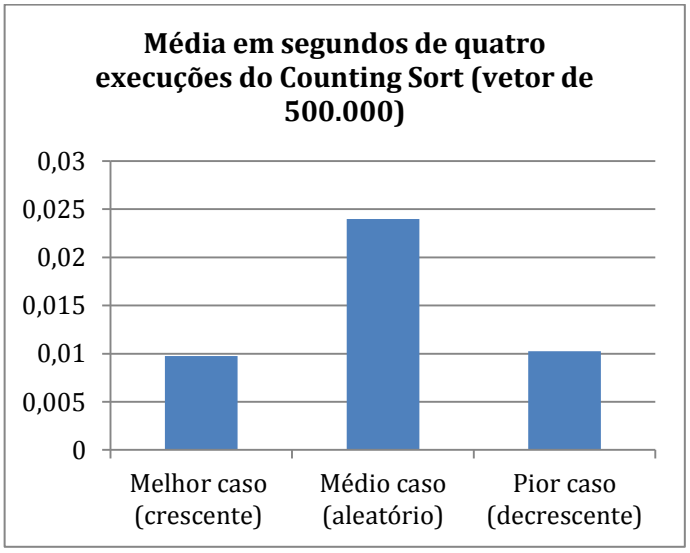
BUCKET SORT			
VETOR	**Melhor caso(cresc)**	**Médio caso (aleat)**	**Pior caso (decresc)**
50.000:	0,00375	0,0125	0,005
100.000:	0,00875	0,026	0,01025
500.000:	0,05125	0,1195	0,05525

COUNTING SORT			
VETOR	**Melhor caso(cresc)**	**Médio caso (aleat)**	**Pior caso (decresc)**
50.000:	0,002	0,00395	0,00175
100.000:	0,00575	0,006	0,003
500.000:	0,00975	0,024	0,01025

Para ficar mais nítida a visualização dos resultados geramos um gráfico para cada algoritmo com vetores de 100.000 posições.







6 Conclusão

Neste trabalho foi detalhado um pouco do contexto histórico e das características individuais pertencentes aos algoritmos Merge Sort, Counting Sort e HeapSort a fim de apresentar informações relevantes para a compreensão do funcionamento de cada um. Nos testes, cada algoritmo demonstrou um comportamento diferente, sendo um mais eficiente que outro em certos casos e situações e podendo ser menos eficiente quando o cenário muda de caso e situação. Assim, foi possível notar que cada algoritmo tem eficiência nas suas particularidades, não podendo destacar um algoritmo como melhor que outro em todos os casos. Por fim, fica em aberto ao desenvolvedor escolher, através deste estudo, qual algoritmo resolveria melhor seu problema, dependendo do ambiente em que ele seria aplicado.

O código utilizado para esse artigo está disponível no seguinte Github:

7 Referências bibliográficas

Wikipédia, Bubble sort. Disponivel em: https://pt.wikipedia.org/wiki/Bubble_sort. Acesso em 28 de abril de 2019.

DevFuria, Introdução ao algoritmo de ordenação Bubble Sort. Disponivel em: http://www.devfuria.com.br/logica-de-programacao/introducao-ao-algoritmo-de-ordenacao-bubble-sort/. Acesso em 30 de abril de 2019.

Braga, Henrique. Algoritmos de Ordenação I: Bubble Sort. Disponivel em: https://medium.com/@henriquebraga_18075/algoritmos-de-ordena%C3%A7%C3%A3o-i-bubble-sort-c162a67261ef. Acesso em 30 de abril de 2019.

Artigo Entendendo o Algoritmo Bubble Sort em Java, Dispnível em: https://www.devmedia.com.br/entendendo-o-algoritmo-bubble-sort-em-java/24812. Acesso em 30 de abril de 2019.

Wikipédia, Bucket sort. Disponivel em: https://pt.wikipedia.org/wiki/Bucket_sort. Acesso em 28 de abril de 2019.

GeeksForGeeks, Bucket sort. Disponivel em: https://www.geeksforgeeks.org/bucket-sort-2/. Acesso em 30 de abril de 2019.

Wurthmann, Bucket sort. Disponivel em: http://wurthmann.blogspot.com/2015/06/bucket-sort.html. Acesso em 30 de abril de 2019.

Priuli, Felipe, Ordenação por Counting Sort. Disponível em: https://felipepriuli.wordpress.com/2013/01/08/counting-sort/. Acesso em 18 de abril de 2018.
George Henrique Wurthmann, Bucket sorte. Disponível em: http://wurthmann.blogspot.com/2015/06/bucket-sort.html. Acesso em 30 de abril de 2019.

Wikipédia, Counting sort. Disponivel em: https://pt.wikipedia.org/wiki/Counting_sort. Acesso em: 18 de abril de 2018.

GeeksForGeeks, Counting sort. Disponivel em: https://www.geeksforgeeks.org/counting-sort/. Acesso em: 30 de abril de 2018.
Ojha, Ravi. Counting sort. Disponivel em:

https://www.hackerearth.com/pt-br/practice/algorithms/sorting/counting-sort/tutorial/. Acesso em: 30 de abril de 2018.

Felipe Priuli

Ordenação por Counting Sort. Disponivel em: https://felipepriuli.wordpress.com/2013/01/08/counting-sort/. Acesso em: 18 de abril de 2018.

Counting Sort:

https://www.geeksforgeeks.org/counting-sort/

https://felipepriuli.wordpress.com/2013/01/08/counting-sort/

https://www.hackerearth.com/pt-br/practice/algorithms/sorting/counting-sort/tutorial/

https://pt.wikipedia.org/wiki/Counting_sort