Relatório Estrutura de Dados II Análise do MergeSort

Atividade Prática 1(2020.2)

Kennedy Anderson Mendes Nunes[1]

[1]Universidade Federal do Maranhão (UFMA) – São Luís – MA – Brasil

{kennedy.anderson}@discente.ufma.br

1. Análise de Complexidade

1.1. Algoritmos usados na ordenação.

mergeSortMain() - *class* **MergeSort -** Função responsável por gerenciar o **MergeSort** nele temos a chamadas recursivas e a chamada a função merge(), responsável por executar o MergeSort padrão (ele está na class MergeSort, será reescrito nas outras classes), sua complexidade é *O(n log(n))*

merge() - *class* **MergeSort -** Função responsável por unir os lados dos subvetores separados do MergeSort, funciona como um intercalador das subsequências ordenadas para apresentar a solução. A complexidade é *O(n)*.

mergeSortMain() - *class* **MergeInsert -** Uma função sobrescrita da função **mergeSortMain() -** *class* **MergeSort**, com a diferença do if verificando se os números de elementos é maior que 15, caso contrário é aplicado o **insertSort()**, sua complexidade está em torno de *O(n log(n))*.

insertSort() - *class* **MergeInsert** - Uma função de ordenação por inserção que implementa o método InsertSort, sua complexidade para o melhor caso é O(n), porém se estiver na ordem inversa pode assumir $O(n^2)$.

mergeSortMain() - *class* **MergeSorted -** É a função mergeSortMain - *class* **MergeInsert** sobrescrita, com o acréscimo da verificação se os subvetores está previamente ordenado, como forma de evitar mais custos de execução do algoritmo. Através do CÓDIGO(if (v[meio+1].compareTo(v[meio]) >= 0) return v;), sua complexidade é *O(n log(n))*.

sort() - *class* **MergeX -** Função estáica que faz um clone do vetor auxiliar, complexidade (O n log(n)).

mergeSortMain() - *class* **MergeX -** É a função mergeSortMain - *class* **MergeSorted** sobrescrita, porém com a passagem do vetor é trocada a função recebe o auxiliar como o primeiro vetor, evitando a inicialização e a cópia do vetor, mesmo diminuindo suas operações a complexidade é O(n log(n)).

merge() - *class* **MergeX -** É uma modificação da função merge da *class* **MergeSort** com um clone do vetor, e sem fazer cópia para um auxiliar. Agora em vez de passar o vetor é passado o clone auxiliar com isso conseguimos economizar passos e tempo de execução. A complexidade é *O(n).*

2. Teste

2.1. Descrição

Nos teste foram usados 3 tamanhos de vetores 100, 1000 e 100000 dentre os diferentes tipos gerando um random para os ints de até 1000:

- **TipoS** - **String** - chave, **double** - valor.
- **TipoD** - **double** - chave, **String** - valor
- **TipoInt** - **int** chave, **int** valor

Cada teste da implementação estará nas tabelas abaixo presente na seção 2.2 Resultados a Tabela1, Tabela2 e Tabela3, apresentará o tempo de execução de cada implementação e versão do MergeSort.

2.2. Resultados

As tabelas abaixo irão demonstrar o tempo de execução das diferentes implementações do MergeSort e Arrays.sort do JDK java.

O cálculo desse tempo de execução é usando o currenteTimeMillis() que dá o tempo em milisegundos, após algumas execuções é tirado a média e colocado seu valor aproximado, foi usada a notação cientifica do excel para demonstrar esse valor, onde "E" equivale a base 10.

Tabela1. Comparativo do TipoInt - int chave int valor

Comparativo de Implementação Tipo Int		
Tamanho do Vetor	**Tempo de Execução**	**Implementação**
100	1,00E-04	MergeSort Padrão
100	1,00E-05	Merge Sort com InsertSort
100	1,00E-05	MergeSort verifica a ordenação
100	1,00E-05	MergeSort não copia
100	1,00E-04	Array.sort (Java)
10000	5,00E-03	MergeSort Padrão
10000	3,00E-03	Merge Sort com InsertSort
10000	1,00E-03	MergeSort verifica a ordenação
10000	1,00E-03	MergeSort não copia
10000	1,00E-03	Array.sort (Java)
100000	7,40E-03	MergeSort Padrão
100000	2,40E-03	Merge Sort com InsertSort
100000	1,90E-03	MergeSort verifica a ordenação
100000	1,00E-03	MergeSort não copia

100000	4,00E-03	Array.sort (Java)

Tabela2. Comparativo do TipoD - double chave String valor

Comparativo de Implementação Tipo D		
Tamanho do Vetor	**Tempo de Execução**	**Implementação**
100	1,85E-04	MergeSort Padrão
100	1,55E-05	Merge Sort com InsertSort
100	1,00E-05	MergeSort verifica a ordenação
100	1,00E-05	MergeSort não copia
100	1,62E-04	Array.sort (Java)
1000	3,55E-03	MergeSort Padrão
1000	8,78E-03	Merge Sort com InsertSort
1000	2,45E-04	MergeSort verifica a ordenação
1000	3,16E-03	MergeSort não copia
1000	3,90E-03	Array.sort (Java)
100000	1,58E-02	MergeSort Padrão
100000	4,78E-03	Merge Sort com InsertSort
100000	2,58E-03	MergeSort verifica a ordenação
100000	2,48E-03	MergeSort não copia
100000	7,51E-03	Array.sort (Java)

Tabela2. Comparativo do TipoD - String chave double valor

Comparativo de Implementação Tipo S		
Tamanho do Vetor	**Tempo de Execução**	**Implementação**
100	1,00E-03	MergeSort Padrão
100	1,00E-04	Merge Sort com InsertSort
100	1,00E-05	MergeSort verifica a ordenação
100	1,00E-05	MergeSort não copia
100	1,00E-05	Array.sort (Java)
10000	2,40E-03	MergeSort Padrão
10000	3,00E-03	Merge Sort com InsertSort

10000	2,00E-03	MergeSort verifica a ordenação
10000	2,00E-03	MergeSort não copia
10000	2,00E-03	Array.sort (Java)
100000	7,53E-03	MergeSort Padrão
100000	6,20E-03	Merge Sort com InsertSort
100000	5,00E-03	MergeSort verifica a ordenação
100000	5,00E-03	MergeSort não copia
100000	1,40E-03	Array.sort (Java)

3. Conclusão

Podemos observar o tempo de execução nas classes TipoD (double chave e String Valor) e TipoS(String chave e double valor) é normalmente maior do que a classe TipoInt (int chave int valor), dando destaque ao tipo TipoS devido a seu tamanho com mais de 30 caractere.

A implementação do MergeSort padrão acaba apresentando um custo de tempo maior do que as outras implementações, mas ao fazer a implementação do insertionSort o tempo de execução chega a cair pela metade, esse dado fica mais claro quando é feito a comparação com o vetor tamanho de 100 000.

Com o decorrer das implementações pode ser observado uma queda do tempo de execução de de mais de dois terços do tempo em relação ao tempo de execução do Merge Sort Padrão, isso ocorre devido a redução nas operações da implementação, já que na implementação da *class* **MergeSorted** - ela verifica se dois sub vetores estão ordenados e tem o objetivo economizar algumas operações.

A implementação do MergeSort sem cópia do vetor é se destaca pelo seu baixo custo de tempo de execução, além do seu número de passos de execução pode ser consideravelmente alto. Podemos dar destaque a MergeSort sem Cópia e MergeSort que verifica os sub-vetores ordenados, como mostra o Gráfico1 abaixo seus tempos de execução foram os menores.

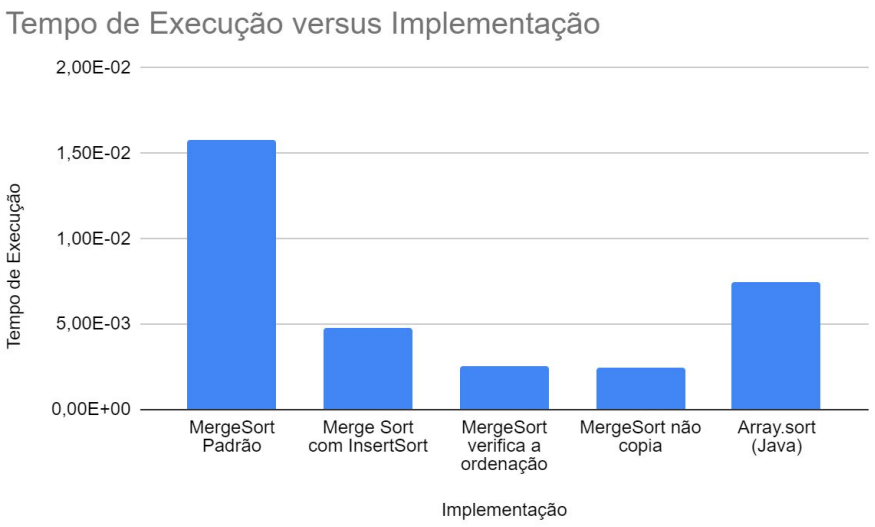

Gráfico 1. Tempo de execução x Implementação (vetor com tamanho 100000)

3.1. Dificuldades Encontradas

Devido ao baixo tempo para a preparação do trabalho e o baixo conhecimento na linguagem Java, minha dificuldade foi a falta de maestria na organização das classes e código. A principal dificuldade que eu encontrei foi a sacada na implementação do MergeSort sem precisar da cópia, além dessa teve a de ler arquivos, já que meu conhecimento Java é baixo demorei 2 dias para conseguir entender o funcionamento das implementações, comecei com o MergeSort Padrão e fui desenvolvendo e crescendo conhecimento no decorrer, devido a falta de tempo o contador de passos não foi implementado, além de falta uma Main bem elaborada dos os testes feitos foram mudados manualmente no código, a minha dificuldade com adicionar comentários deixou difícil a leitura do código.

Tenho plena certeza que dei meu melhor neste trabalho, poderia entregar algo muito bem implementado se tivesse tempo e recurso, mas não se pode começar grande essa implementação foi de grande ajuda na minha caminhada como programador, aprendi muito, mesmo não implementado tão bem, sei que o aprendizado ficará e o conhecimento jamais irá se expirar.

4. Links

Vídeo -

GitHub - https://github.com/kennedy9616/MergeSort.git

5. Referências

SEDGEWICK, Robert & WAYNE Kevin . Algorithms. 2.2 MERGESORT. Disponível em: https://algs4.cs.princeton.edu/home/. Acesso em: março de 2021.

DevMedia. Lendo dados de Txt com Java. Disponível em: https://www.devmedia.com.br/lendo-dados-de-txt-com-java/23221. Acesso em março de 2021.

Java SE. OpenJDK Wiki. Wiki com as principais funções java. Disponível em: https://pt.stackoverflow.com/. Acesso em: março de 2021

DevMedia. Algoritmos de ordenação. Disponível em: https://www.devmedia.com.br/algoritmos-de-ordenacao/2622. Acesso em: março 2021