UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

João Vitor da Silva e Diego Ribeiro Chaves

IMPLEMENTAÇÃO DE ALGORITMOS HÍBRIDOS DE ORDENAÇÃO

Santa Maria, RS
2024

RESUMO

IMPLEMENTAÇÃO DE ALGORITMOS HÍBRIDOS DE ORDENAÇÃO

AUTORES: João Vitor da Silva e Diego Ribeiro Chaves
PROFESSOR: Antônio Marcos de Oliveira Candia

Este relatório aborda a implementação dos algoritmos híbridos de ordenação QuickInsert e MergeInsert. O objetivo é consolidar o conhecimento em técnicas de ordenação de dados. Os algoritmos foram avaliados em situações de ordenação interna e externa, e o relatório conclui com uma análise das situações em que cada algoritmo é mais eficaz.

Palavras-chave: Algoritmos de Ordenação. QuickInsert. MergeInsert. Ordenação Interna e Externa. Desempenho de Algoritmos. Comparação de Algoritmos

SUMÁRIO

1 INTRODUÇÃO

Este relatório apresenta um estudo prático sobre técnicas de ordenação de dados, com foco na implementação e avaliação de dois algoritmos híbridos de ordenação: QuickInsert e MergeInsert. O objetivo deste trabalho é consolidar o conhecimento teórico através da aplicação prática, permitindo uma compreensão mais profunda dos princípios e desafios associados à ordenação de dados.

Os algoritmos QuickInsert e MergeInsert foram escolhidos por sua eficácia em diferentes cenários de ordenação. O QuickInsert combina a eficiência do Quicksort em grandes conjuntos de dados com a simplicidade e eficiência do algoritmo de inserção em conjuntos menores. De forma semelhante, o MergeInsert combina o Mergesort com o algoritmo de inserção, proporcionando uma solução robusta e eficiente para a ordenação de dados.

Este relatório detalha a estrutura do programa, incluindo a função principal e várias funções auxiliares. Além disso, descreve o funcionamento dos algoritmos de ordenação híbrida e apresenta uma análise comparativa do desempenho dos algoritmos em situações de ordenação interna e externa.

2 ALGORITMO QUICKINSERT

O algoritmo QuickInsert é uma técnica de ordenação híbrida que combina Quicksort e Inserção. Utiliza o Quicksort para grandes conjuntos e Inserção para menores, otimizando assim o processo de ordenação.

2.1 OBJETIVO

O programa tem como objetivo ordenar um conjunto de valores contidos em um arquivo de texto utilizando um algoritmo de ordenação híbrida. Ele combina as vantagens do Quicksort, que é eficiente em grandes conjuntos de dados, com a simplicidade e eficiência do algoritmo de Ordenação por Inserção em conjuntos menores.

2.2 ESTRUTURA DO PROGRAMA

O programa é estruturado em torno da função principal *main* e várias funções auxiliares:

- **main** : A função principal é responsável por coordenar a execução do programa, lendo o arquivo, chamando os algoritmos e imprimindo o vetor ordenado.
- **swap** : Essa função simplesmente troca dois elementos de um vetor.
- **ordenacaoInsercao**: Implementa o algoritmo de ordenação por inserção.
- **particionar**: Divide o vetor em torno de um pivô para aplicar o algoritmo de ordenação quicksort.
- **ordenacaoHibrida**: Implementa o algoritmo de ordenação híbrida, decidindo entre o quicksort e o inserção dependendo do tamanho do vetor.

2.3 FUNCIONAMENTO

O algoritmo de ordenação híbrida implementado neste programa combina as vantagens do Quicksort e da ordenação por inserção para obter um desempenho otimizado em diferentes tamanhos de conjuntos de dados. Aqui está uma explicação detalhada do seu funcionamento:

2.3.1 Divisão e Conquista

O algoritmo inicia com a divisão do vetor em partes menores para a ordenação. Se o tamanho da parte do vetor a ser ordenada for menor do que um limite definido (LIMITAR), o algoritmo opta por usar a ordenação por inserção, que é eficiente para conjuntos pequenos de dados.

2.3.2 Escolha do Algoritmo de Ordenação

Quando o tamanho da parte do vetor é maior do que o limite definido, o algoritmo escolhe usar o Quicksort para ordenar essa parte. Ele seleciona um elemento pivô e rearranja o vetor de modo que todos os elementos menores que o pivô estejam à esquerda e todos os elementos maiores estejam à direita. Em seguida, o algoritmo divide o vetor em duas partes ao redor do pivô e chama recursivamente a função de ordenação híbrida para cada parte.

2.3.3 Estratégia de Recursão

Durante a recursão, o algoritmo continua dividindo as partes do vetor em partes menores e escolhendo o método de ordenação apropriado com base no tamanho dessas partes. Se uma parte do vetor se tornar pequena o suficiente durante a recursão, o algoritmo usará a ordenação por inserção para finalizar a ordenação dessa parte, evitando assim o overhead de continuar a dividir e classificar ainda mais.

2.3.4 Ordenação das Partes Restantes

Após ordenar uma parte do vetor usando Quicksort ou ordenação por inserção, o algoritmo repete o processo para as partes restantes do vetor até que todas as partes estejam ordenadas. Esse processo de divisão e ordenação continua até que o vetor inteiro seja ordenado.

2.3.5 Vantagens da Abordagem Híbrida

A abordagem híbrida oferece uma vantagem significativa em termos de desempenho, combinando a eficiência do Quicksort em grandes conjuntos de dados com a simplici-

dade e eficiência da ordenação por inserção em conjuntos menores. Ela adapta dinamicamente o algoritmo de ordenação com base no tamanho do conjunto de dados, resultando em um desempenho otimizado em uma ampla gama de cenários.

2.4 RESULTADOS

O programa é capaz de ler um conjunto de dados de um arquivo, ordená-lo utilizando o algoritmo de ordenação híbrida e imprimir o resultado ordenado na saída padrão. O resultado é uma lista ordenada dos valores presentes no arquivo, com a garantia de que está corretamente ordenada de forma crescente.

3 ALGORITMO MERGEINSERT

O algoritmo MergeInsert é uma técnica de ordenação híbrida que combina MergeSort e Inserção. Utiliza o MergeSort para grandes conjuntos e Inserção para menores, otimizando assim o processo de ordenação.

3.1 OBJETIVO

O programa tem como objetivo ordenar um conjunto de valores contidos em um arquivo de texto utilizando um algoritmo de ordenação híbrida. Ele combina as vantagens do Mergesort, que é eficiente em grandes conjuntos de dados, com a simplicidade e eficiência do algoritmo de Ordenação por Inserção em conjuntos menores.

3.2 ESTRUTURA DO PROGRAMA

O programa é estruturado em torno da função principal *main* e várias funções auxiliares:

- **main** : A função principal é responsável por coordenar a execução do programa, lendo o arquivo, chamando os algoritmos e imprimindo o vetor ordenado.
- **insertionSort** : A função implementa o algoritmo de ordenação por inserção, e ordena parcialmente o vetor antes de chamar a função que implementa o Merge Sort.
- **merge**: Implementa o algoritmo de inserção Merge Sort.
- **particionar**: Divide o vetor em torno de um pivô para aplicar o algoritmo de ordenação quicksort.
- **hybridSort**: Implementa o algoritmo de ordenação híbrido, que dependendo do tamanho do vetor utiliza o Merge ou o Insertion Sort.

3.3 FUNCIONAMENTO

O algoritmo de ordenação híbrida implementado neste programa combina as vantagens do Mergesort e da ordenação por inserção para obter um desempenho otimizado em diferentes tamanhos de conjuntos de dados. Aqui está uma explicação detalhada do seu funcionamento:

3.3.1 Estratégia Geral]

O algoritmo de ordenação híbrida divide o array recursivamente até que a sublista seja pequena o suficiente para ser ordenada eficientemente pelo insertion sort. Caso a sublista seja maior que um limiar definido, o merge sort é aplicado. Esta abordagem combina a eficiência do insertion sort em listas pequenas com a eficiência do merge sort em listas maiores.

3.3.2 Algoritmo de Ordenação por Inserção (Insertion Sort)

Utilizado para ordenar subconjuntos pequenos do array. Esse algoritmo percorre a sublista e insere cada elemento na posição correta, mantendo o restante da sublista ordenada. É eficiente em conjuntos de dados pequenos devido à sua complexidade de tempo O(n^2).

3.3.3 Algoritmo de Ordenação Merge

O merge sort divide o array em duas metades, ordenando recursivamente cada metade, tendo boa eficiência em grandes conjuntos de dados, pois garante a complexidade de tempo O(n log n) em todos os casos. Ao final, intercala as duas metades para produzir o array final.

3.3.4 Combinação dos Algoritmos

O algoritmo híbrido combina os algoritmos de insertion sort e merge sort, aproveitando suas vantagens em diferentes contextos. A escolha entre os algoritmos é feita com base em um limiar, definido pelo programador, nesse caso utilizamos o 10. Isso significa que enquanto o array tiver tamanho maior que o limiar, o algoritmo usado para ordenação será o Merge, por ter maior eficiência em conjuntos de dados maiores, mas ao termos um array menor que o limiar, utilizamos o Insertion, por se dar melhor com poucos dados.

3.4 RESULTADOS

O programa é capaz de ler um conjunto de dados de um arquivo, ordená-lo utilizando o algoritmo de ordenação híbrida e imprimir o resultado ordenado na saída padrão.

O resultado é uma lista ordenada dos valores presentes no arquivo, com a garantia de que está corretamente ordenada de forma crescente.

4 COMPARAÇÃO

A partir dos testes que contabilizam o tempo de execução de cada algoritmo em diferentes cenários de ordenação, temos:

Vetor/Tempo Algoritmo	QuickInsert	MergeInsert
interno aleatorio	0,002s	0,002s
interno quase ordenado	0,002s	0,001s
interno inverso	0,000s	0,000s
externo aleatorio	0,046s	0,074s
externo quase ordenado	0,243s	0,045s
externo inverso	0,032s	0,042s

Em uma analise rápida, concluímos que o algoritmo QuickInsert é ligeiramente mais rápido em cenários de ordenação interna inversa e externa aleatória, indicando uma melhor performance em dados não ordenados. Já o MergeInsert mostra uma eficiência superior em cenários de ordenação externa, especialmente quando os dados estão quase ordenados, sugerindo uma melhor adaptação a dados parcialmente ordenados.

5 CONCLUSÃO

O QuickInsert oferece uma estratégia eficiente para a ordenação de dados, pois combina dois algoritmos bons em diferentes situações. O Quicksort é geralmente mais eficiente em grandes conjuntos de dados devido à sua complexidade média de tempo de execução de O(n log n), onde "n"é o número de elementos no conjunto de dados. Ele é um algoritmo de "dividir para conquistar"que divide o conjunto de dados em partições menores, ordena recursivamente essas partições e, em seguida, combina as partições ordenadas. Isso resulta em uma eficiência considerável, especialmente para grandes conjuntos de dados.

Por outro lado, a ordenação por inserção tem um desempenho melhor em conjuntos de dados menores. Apesar de ter uma complexidade de tempo de execução de O(n^2) no pior caso, é um algoritmo simples e eficiente para conjuntos de dados pequenos. Ele funciona comparando cada elemento com os elementos anteriores no vetor e inserindo-o na posição correta, resultando em uma execução mais rápida em conjuntos menores, onde o overhead de divisão e recursão do Quicksort pode ser desnecessário.

A estratégia de ordenação híbrida neste programa combina esses dois algoritmos, aproveitando as vantagens de cada um deles: o Quicksort para grandes conjuntos de dados e a ordenação por inserção para conjuntos menores, resultando em um desempenho otimizado para uma variedade de tamanhos de conjuntos de dados.

Já a implementação de um algoritmo MergeInsert de ordenação híbrida apresenta uma abordagem eficaz para lidar com a ordenação de conjuntos de dados em diferentes contextos. Ao combinar os algoritmos de insertion sort e merge sort, o programa consegue adaptar-se eficientemente a diferentes tamanhos de conjuntos de dados, proporcionando uma solução versátil e eficiente para problemas de ordenação. A flexibilidade e eficácia desse algoritmo o tornam uma escolha viável para aplicações que exigem ordenação rápida e eficiente de grandes conjuntos de dados.