Resolução de Problemas Estruturados em Computação

Mateus M. Batista[1]

[1]Escola Politecnica – Pontifícia Universidade Católica do Paraná (PUCPR)
Rua Imaculada Conceição, 1155 – Bairro Prado Velho – CEP 80215-901

Abstract. *Effective student work related to the computer science course on structured problem solving in computing.*

Resumo. *Trabalho discente efetivo referente ao curso de ciência da computação de resolução de problemas estruturados em computação.*

1. Informações

O relatório a seguir corresponde a análise de resultados de implementações de ordenações, sendo shell sort, quick sort e bubble sort

2. Bubble Sort

Um array chamado tamanhos é definido, que contém os tamanhos dos vetores a serem testados.

A variável numTestes define quantos testes serão realizados para cada tamanho de vetor.

O programa entra em um loop que itera sobre os tamanhos dos vetores especificados.

Para cada tamanho de vetor, um novo vetor é gerado aleatoriamente usando a função gerarVetorAleatorio().

O algoritmo QuickSort é aplicado ao vetor gerado, e o tempo de execução, o número de trocas e o número de iterações são registrados.

Os resultados médios dos testes (tempo de execução médio, trocas médias e iterações médias) para o tamanho do vetor atual são calculados.

Os resultados são impressos no console, incluindo o tamanho do vetor, o tempo de execução médio, o número médio de trocas e o número médio de iterações.

O programa passa para o próximo tamanho do vetor e repete o processo.

A classe ArrayBub possui três campos de dados:

a: Um array de números inteiros que será usado para armazenar os elementos a serem ordenados. nElems: Um inteiro que acompanha o número de elementos presentes no array. O construtor ArrayBub é usado para criar uma instância da classe e inicializar o array a com o tamanho especificado (definido pelo argumento max) e nElems com 0, indicando que inicialmente não há elementos no array.

O método insert(long value) permite adicionar elementos ao array. Ele coloca o valor value na próxima posição disponível no array a e, em seguida, incrementa nElems para indicar que um novo elemento foi inserido.

Bubble Sort			
tamanho	tempo(nanosegundos)	trocas	iterações
50	45359	623	1198
500	1492280	63639	124439
1000	1788020	248163	498814
50000	25881060	6268271	12496285
10000	128755960	24977251	49981790

Figure 1. Tabela resultante operações de Bubble Sort

O método display() é usado para imprimir o conteúdo do array na saída padrão (geralmente, a tela). Ele percorre o array e imprime cada elemento seguido de um espaço em branco, seguido de uma quebra de linha no final.

O método bubbleSort() implementa o algoritmo Bubble Sort para ordenar os elementos no array. O Bubble Sort é um algoritmo de ordenação simples que funciona comparando pares de elementos adjacentes e trocando-os se estiverem fora de ordem. Isso é repetido até que nenhum elemento precise ser trocado em uma passagem completa. O loop externo (for (out = nElems - 1; out ¿ 1; out–)) controla as passagens, e o loop interno (for (in = 0; in ¡ out; in++)) compara e troca os elementos.

O método privado swap(int one, int two) é usado para trocar dois elementos no array. É uma função auxiliar chamada pelo bubbleSort() quando é necessário trocar dois elementos de posição.

3. Quick Sort

Método main: O método main é onde o programa começa a ser executado. Ele contém um loop que itera através de diferentes tamanhos de arrays (50, 500, 1000, 5000 e 10000) especificados no array tamanhos.

Loop Externo: O loop externo percorre os diferentes tamanhos de arrays em tamanhos.

Variáveis de Estatísticas: Para cada tamanho de array, você inicializa três variáveis de long (inteiros longos) para rastrear o tempo de execução total, o total de trocas e o total de iterações. Essas variáveis são tempoExecucaoTotal, totalTrocas e totalIteracoes.

Loop Interno: Há um loop interno que realiza testes para o mesmo tamanho de array. O número de testes é definido em numTestes, que é igual a 5. Portanto, o código dentro do loop interno será executado 5 vezes para cada tamanho de array.

Método gerarVetorAleatorio: Esse método é responsável por gerar um array de números inteiros aleatórios do tamanho especificado. Ele usa a classe Random para gerar os números aleatórios.

Mensuração do Tempo de Execução do Quick Sort: O tempo de execução do algoritmo Quick Sort é medido usando System.nanoTime(). O tempo é registrado antes e depois da chamada ao método quicksort para ordenar o array.

Chamada do Método quicksort: O método QuickSort.quicksort é chamado para ordenar o array aleatório gerado. Ele retorna um objeto QuickSort.ResultadoQuickSort, que contém informações sobre o número de trocas e iterações realizadas pelo algoritmo.

Cálculo das Estatísticas Médias: Após cada teste, o código atualiza as variáveis tempoExecucaoTotal, totalTrocas e totalIteracoes com os resultados do teste.

Cálculo das Médias: Após todos os testes para um determinado tamanho de array, as médias de tempo de execução, trocas e iterações são calculadas dividindo os totais pelos números de testes.

Exibição de Resultados: O programa exibe as estatísticas médias, incluindo o tamanho do array, tempo de execução médio (em milissegundos), trocas médias e iterações médias para cada tamanho de array.

Repetição para Outros Tamanhos de Array: O loop externo continua para os próximos tamanhos de array.

Classe ResultadoQuickSort: Esta classe é usada para encapsular os resultados do algoritmo Quick Sort, ou seja, o número de trocas e iterações feitas durante a ordenação.

Método quicksort: Este é o método público que inicia o processo de ordenação. Ele recebe um array de inteiros como entrada e retorna um objeto ResultadoQuickSort. O método cria um array trocasIteracoes para rastrear o número de trocas e iterações. Em seguida, chama o método quicksort privado para realizar a ordenação do array.

Método privado quicksort: Este é o método de ordenação principal. Ele recebe o array, o índice de início e o índice de fim, bem como o array trocasIteracoes. O algoritmo Quick Sort é recursivamente aplicado a sub-arrays até que o array seja completamente ordenado. A ordenação é feita da seguinte forma:

Verifica se inicio é menor do que fim para garantir que haja mais de um elemento a ser ordenado. Chama o método privado particiona para determinar o índice do pivo e dividir o array em duas partes. Chama recursivamente quicksort para ordenar as duas partes separadamente. Método privado particiona: Este método é responsável por escolher um elemento pivo no array e rearranjar os elementos de modo que os menores que o pivo fiquem à esquerda e os maiores à direita. Ele recebe o array, o índice de início e o índice de fim, bem como o array trocasIteracoes. O método realiza o seguinte:

Define o pivo como o primeiro elemento do array (array[inicio]) e o índice i como inicio. Itera através do array a partir do segundo elemento (j = inicio + 1) até o último elemento (fim). Incrementa o contador de iterações. Se um elemento for menor que o pivo, ele é trocado com o elemento na posição i + 1, e o contador de trocas é incrementado. Isso garante que os elementos menores que o pivo fiquem à esquerda. No final da iteração, o pivo é trocado com o elemento na posição i, garantindo que o pivo esteja na posição correta. Retorna o índice i como o novo índice do pivo. No método main da classe App, você gera arrays aleatórios de diferentes tamanhos, chama o método quicksort para ordená-los e registra o tempo de execução médio, o número médio de trocas e o número médio de iterações para cada tamanho de array.

4. ShellSort

Classe App: Esta classe serve como o ponto de entrada do programa.

QuickSort			
tamanho	tempo(ms)	trocas	iterações
50	0.68548	137.2	203
500	0.58822	3054.8	3506
1000	0.67482	7574	8006
50000	1,92574	60965	55005
10000	3,23858	148098	120005

Figure 2. Tabela resultante operações Quick Sort

Método main: O método main é onde o programa começa a ser executado. Ele contém um loop que itera através de diferentes tamanhos de arrays especificados no array tamanhos (50, 500, 1000, 5000, 10000).

Loop Externo: O loop externo percorre os diferentes tamanhos de arrays em tamanhos.

Variáveis de Estatísticas: Para cada tamanho de array, você inicializa três variáveis de long (inteiros longos) para rastrear o tempo de execução total, o total de trocas e o total de iterações. Essas variáveis são tempoExecucaoTotal, totalTrocas e totalIteracoes.

Loop Interno: Há um loop interno que realiza testes para o mesmo tamanho de array. O número de testes é definido em numTestes, que é igual a 5. Portanto, o código dentro do loop interno será executado 5 vezes para cada tamanho de array.

Método gerarArrayAleatorio: Este método é responsável por gerar um array de números inteiros aleatórios do tamanho especificado. Ele usa a classe Random para gerar os números aleatórios.

Mensuração do Tempo de Execução do Shell Sort: O tempo de execução do algoritmo Shell Sort é medido usando System.nanoTime(). O tempo é registrado antes e depois da chamada ao método shellSort para ordenar o array.

Chamada do Método shellSort: O método ShellSort.shellSort é chamado para ordenar o array aleatório gerado. Ele retorna um objeto ShellSort.ResultadoShellSort, que contém informações sobre o número de trocas e iterações realizadas pelo algoritmo.

Cálculo das Estatísticas Médias: Após cada teste, o código atualiza as variáveis tempoExecucaoTotal, totalTrocas e totalIteracoes com os resultados do teste.

Cálculo das Médias: Após todos os testes para um determinado tamanho de array, as médias de tempo de execução, trocas e iterações são calculadas dividindo os totais pelos números de testes.

Exibição de Resultados: O programa exibe as estatísticas médias, incluindo o tamanho do array, tempo de execução médio (em milissegundos), trocas médias e iterações médias para cada tamanho de array.

Classe ResultadoShellSort: Esta classe é usada para encapsular os resultados do algoritmo Shell Sort, ou seja, o número de trocas e iterações feitas durante a ordenação.

ShellSort			
tamanho	tempo(ms)	trocas	iterações
50	0.31362	170	203
500	0.34094	3109,6	3506
1000	0.50912	7614.8	8006
50000	1,37882	61605.8	55005.0
10000	6,80406	148799.2	120005.0

Figure 3. Tabela resultante operações Shell Sort

Método shellSort: Este é o método público que realiza a ordenação do array. Ele recebe um array de inteiros como entrada e retorna um objeto ResultadoShellSort. O método inicia a ordenação do array usando o algoritmo Shell Sort.

Variáveis Locais: Dentro do método shellSort, você declara três variáveis locais: n, que representa o tamanho do array, e trocas e iteracoes, que são inicializadas como zero e usadas para rastrear o número de trocas e iterações realizadas.

Laços Aninhados: O algoritmo Shell Sort usa dois laços aninhados para realizar a ordenação:

O primeiro laço for externo controla o tamanho do ”gap”, que é iniciado com n / 2 e é reduzido pela metade a cada iteração até que o gap seja igual a 1. O segundo laço for interno itera pelo array a partir do elemento gap até o final do array. Dentro do segundo laço, o algoritmo realiza a ordenação do subarray usando a técnica de inserção. Técnica de Inserção: A técnica de ordenação usada é semelhante à ordenação por inserção. Para cada elemento vetor[i], o algoritmo compara vetor[i] com os elementos vetor[i - gap], vetor[i - 2 * gap], e assim por diante, enquanto o elemento atual for menor do que o elemento comparado. O algoritmo move os elementos maiores à direita e insere o elemento temp na posição correta.

Contagem de Trocas e Iterações: O algoritmo rastreia o número de trocas e iterações feitas durante a ordenação. A cada troca, incrementa a variável trocas, e a cada iteração do laço mais interno, incrementa a variável iteracoes.

ResultadoShellSort: Após a ordenação completa do array, os valores de trocas e iteracoes são encapsulados em um objeto ResultadoShellSort e retornados como resultado da ordenação.