Estudo sobre a escolha do pivô para Quicksort

Antônio Drumond

Instituto de Ciências Exatas e Informática

Ciências da Computação

Felipe Domingos da Cunha

30/09/2024

Introdução

O objetivo deste relatório é analisar e comparar a performance de quatro diferentes estratégias de escolha do pivô no algoritmo de ordenação Quicksort: Pivô no início do arranjo, pivô no final do arranjo, escolha aleatória do pivô, e Pivô na mediana de três elementos.

Essa análise é relevante pois, apesar de o Quicksort ser conhecido por sua complexidade de execução $\mathbf{\Theta}(n*\lg(n))$, em seu pior caso, esse algoritmo pode alcançar $O(n^2)$, e esse pior caso ocorre quando o pivô escolhido é, sistematicamente o maior ou menor elemento no arranjo (Ou na parcela analisada pela chamada recursiva). Portanto, é essencial para manter o custo computacional baixo escolher uma estratégia que minimize a chance de o pivô ser o maior ou menor elemento.

Estudo sobre a escolha do pivô para Quicksort

Agora, serão explicados cada um dos métodos de escolha do pivô e os códigos em Java utilizados em cada um dos algoritmos. Por fim, serão apresentados os resultados dos testes práticos de tempo de execução.

Pivô no início do arranjo

Esse método consiste em utilizar como pivô o elemento mais à “esquerda” no arranjo em questão, ou seja: o valor na posição “left” passado à chamada recursiva.

```
static void QuickSortFirstPivot(int[] arr, int left, int right){
	int i = left,
		j = right;
	int pivo = arr[left];
	while(i<=j){
		while(arr[i]<pivo)
		i++;
		while(arr[j]>pivo)
		j--;
		if(i<=j){
		swap(arr, i, j);
		i++;
		j--;
		}
	}
	if(left<j)
		QuickSortFirstPivot(arr, left, j);
	if(i<right)
		QuickSortFirstPivot(arr, i, right);
}
```

Pivô no final do arranjo

Esse método consiste em utilizar como pivô o elemento mais à “direita” no arranjo em questão, ou seja: o valor na posição “right” passado à chamada recursiva.

```
static void QuickSortLastPivot(int[] arr, int left, int right){
	int i = left,
		j = right;
	int pivo = arr[right];
	while(i<=j){
		while(arr[i]<pivo)
		i++;
		while(arr[j]>pivo)
		j--;
		if(i<=j){
		swap(arr, i, j);
		i++;
		j--;
		}
	}
	if(left<j)
		QuickSortLastPivot(arr, left, j);
	if(i<right)
		QuickSortLastPivot(arr, i, right);
}
```

Pivô em posição aleatória

Esse método consiste em utilizar como pivô um elemento qualquer do arranjo, escolhido em uma posição entre “left” e “right”, utilizando a classe nativa “Random”.

No código em questão, um objeto da classe “Random” global à classe principal foi utilizado para gerar os números aleatórios. Esse objeto foi chamado de “pivoR”

```
static void QuickSortRandomPivot(int[] arr, int left, int right){
	int i = left,
		j = right;
	int pivo = arr[pivoR.nextInt(right-left) + left];
	while(i<=j){
		while(arr[i]<pivo)
		i++;
		while(arr[j]>pivo)
		j--;
		if(i<=j){
		swap(arr, i, j);
		i++;
		j--;
		}
	}
	if(left<j)
		QuickSortRandomPivot(arr, left, j);
	if(i<right)
		QuickSortRandomPivot(arr, i, right);
}
```

Pivô na mediana de três elementos

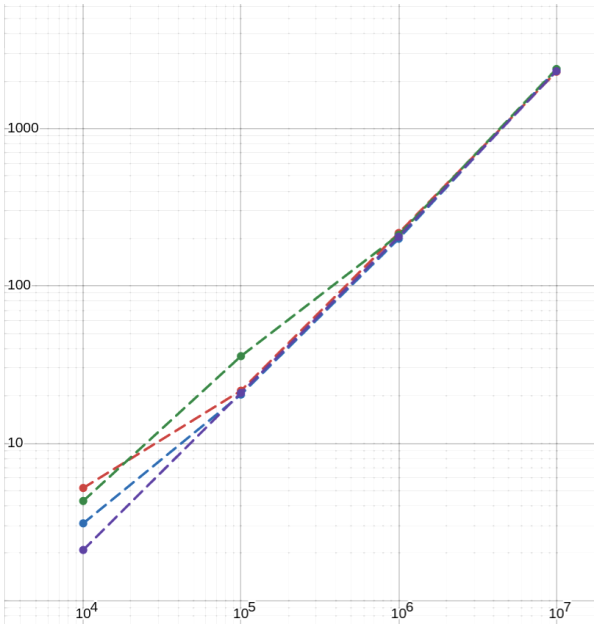
Esse método consiste em calcular a mediana de três elementos do arranjo: As extremidades da esquerda, direita, e o meio. Essa estratégia visa trazer ao mínimo a chance de escolher o menor ou maior elemento do arranjo, algo essencial quando ordenando grandes conjuntos.

```
static int getMedian(int[] arr, int a, int b, int c){
	int[] tmp = new int[3];
	tmp[0] = arr[a];
	tmp[1] = arr[b];
	tmp[2] = arr[c];
	if(tmp[0]>tmp[1]) swap(tmp, 0, 1);
	if(tmp[1]>tmp[2]) swap(tmp, 1, 2);
	if(tmp[0]>tmp[1]) swap(tmp, 0, 1);
	return tmp[2];
	}

	static void QuickSortMedianOfThree(int[] arr, int left, int right){
	int i = left,
		j = right;
	int pivo = getMedian(arr, left, right, (left+right)/2);
	while(i<=j){
		while(arr[i]<pivo)
		i++;
		while(arr[j]>pivo)
		j--;
		if(i<=j){
		swap(arr, i, j);
		i++;
		j--;
		}
	}
	if(left<j)
		QuickSortLastPivot(arr, left, j);
	if(i<right)
		QuickSortLastPivot(arr, i, right);
	}
```

Testes práticos

Após realizar testes práticos em Java, ordenando arranjos de diferentes tamanhos com os quatro algoritmos diferentes, cheguei à conclusão que, em arranjos desordenados com uma desorganização irregular e aleatória, a escolha do pivô não afeta significativamente o tempo de execução do Quicksort. Caso os arranjos de teste estivessem, por exemplo, preenchidos na ordem decrescente, os métodos que escolhem o pivô no início ou final do arranjo teriam complexidade $\Theta(n^2)$, pois os pivôs escolhidos seriam sempre o maior ou menor elementos do arranjo. No entanto, isso não pôde ser observado em meus testes, pois estes foram feitos com arranjos preenchidos aleatoriamente.



	10^4	10^5	10^6	10^7
Primeiro elemento	5.2	21.6	217.1	2300.3
Ultimo elemento	3.1	20.4	199.5	2381.8
Elemento aleatorio	4.3	35.8	211.1	2390.8
Mediana de tres	2.1	20.9	205.5	2317.2

Discussão

Com o objetivo de otimizar o Quicksort, a escolha do pivô não tem um impacto significativo no tempo de execução em arranjos com elementos distribuidos uniformemente. No entanto, a performance do código pode ser melhorada com uma estratégia híbrida, utilizando o Quicksort nas maiores parcelas de elementos (30+ elementos), e chamando o Insertion Sort nas parcelas menores, visto que ele é melhor aplicado em conjuntos já quase ordenados.