Implementação do QuickSort com Variações na Escolha do Pivô

Objetivo

O objetivo deste trabalho é implementar o algoritmo QuickSort variando a estratégia de escolha do pivô e entender como essas variações afetam o desempenho do algoritmo. As quatro estratégias que serão abordadas são: usar o primeiro elemento como pivô, o último elemento, um pivô aleatório, e a mediana de três elementos (início, meio e fim). Após as implementações, será feita uma análise de desempenho com vetores de diferentes tamanhos e características (ordenados, quase ordenados e aleatórios).

Estratégias de Escolha do Pivô

1. Primeiro Elemento como Pivô

Esta estratégia usa o primeiro elemento do vetor como pivô. O algoritmo funciona da seguinte forma:

- O pivô é o primeiro elemento do vetor.
- Após a partição, o pivô estará na posição correta.
- O QuickSort é chamado recursivamente nas duas subpartições.

Implementação:

```
void QuickSortPrimeiroPivo(int vetor[], int esquerda, int direita) {
    if (esquerda < direita) {
        int indicePivo = esquerda;  // Pivô é o primeiro elemento
        int novoIndicePivo = Particionar(vetor, esquerda, direita, indicePivo);
        QuickSortPrimeiroPivo(vetor, esquerda, novoIndicePivo - 1);
        QuickSortPrimeiroPivo(vetor, novoIndicePivo + 1, direita);
    }
}
```

2. Último Elemento como Pivô

Nesta estratégia, o pivô escolhido é o último elemento do vetor. O comportamento do algoritmo é semelhante à versão que usa o primeiro elemento, mas o pivô é o último.

Implementação:

```
void QuickSortUltimoPivo(int vetor[], int esquerda, int direita) {
    if (esquerda < direita) {
        int indicePivo = direita;  // Pivô é o último elemento
```

```
        int novoIndicePivo = Particionar(vetor, esquerda, direita, indicePivo);
        QuickSortUltimoPivo(vetor, esquerda, novoIndicePivo - 1);
        QuickSortUltimoPivo(vetor, novoIndicePivo + 1, direita);
    }
}
```

3. Pivô Aleatório

Nesta variação, o pivô é escolhido aleatoriamente dentro do intervalo de elementos do vetor. Essa estratégia pode ajudar a evitar casos patológicos em que o vetor já está ordenado.

Implementação:

```
#include <stdlib.h>  // Para a função rand()

void QuickSortPivoAleatorio(int vetor[], int esquerda, int direita) {
    if (esquerda < direita) {
        int indicePivo = esquerda + rand() % (direita - esquerda + 1);  // Pivô aleatório
        int novoIndicePivo = Particionar(vetor, esquerda, direita, indicePivo);
        QuickSortPivoAleatorio(vetor, esquerda, novoIndicePivo - 1);
        QuickSortPivoAleatorio(vetor, novoIndicePivo + 1, direita);
    }
}
```

4. Mediana de Três Elementos (Início, Meio e Fim)

Esta estratégia escolhe o pivô como a mediana de três elementos: o primeiro, o elemento do meio e o último. A mediana de três é uma técnica popular para evitar os piores casos de QuickSort (por exemplo, quando o vetor já está ordenado ou quase ordenado).

Implementação:

```
int MedianaDeTres(int vetor[], int esquerda, int direita) {
    int meio = (esquerda + direita) / 2;
    if (vetor[esquerda] > vetor[meio])
        Trocar(&vetor[esquerda], &vetor[meio]);
    if (vetor[esquerda] > vetor[direita])
        Trocar(&vetor[esquerda], &vetor[direita]);
    if (vetor[meio] > vetor[direita])
        Trocar(&vetor[meio], &vetor[direita]);

    return meio;  // O índice da mediana
}

void QuickSortMedianaTres(int vetor[], int esquerda, int direita) {
    if (esquerda < direita) {
        int indicePivo = MedianaDeTres(vetor, esquerda, direita);  // Pivô é a mediana de três
```

```
        int novoIndicePivo = Particionar(vetor, esquerda, direita, indicePivo);
        QuickSortMedianaTres(vetor, esquerda, novoIndicePivo - 1);
        QuickSortMedianaTres(vetor, novoIndicePivo + 1, direita);
    }
}
```

Funções Auxiliares

Todas as versões do QuickSort utilizam a função de particionamento e a função de troca, que são responsáveis por reorganizar o vetor com base no pivô e trocar elementos.

Implementação das Funções Auxiliares

```
void Trocar(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int Particionar(int vetor[], int esquerda, int direita, int indicePivo) {
    int pivo = vetor[indicePivo];
    Trocar(&vetor[indicePivo], &vetor[direita]);  // Move o pivô para o final
    int indiceParticao = esquerda;

    for (int i = esquerda; i < direita; i++) {
        if (vetor[i] <= pivo) {
            Trocar(&vetor[i], &vetor[indiceParticao]);
            indiceParticao++;
        }
    }
    Trocar(&vetor[indiceParticao], &vetor[direita]);  // Move o pivô para sua posição correta
    return indiceParticao;
}
```

Análise de Desempenho

Configuração do Teste

Para avaliar o desempenho das diferentes estratégias de escolha de pivô, executei cada versão do QuickSort com vetores de tamanhos diferentes (100, 1.000, e 10.000 elementos). Além disso, testei três tipos de vetores:

- **Vetor Ordenado**: Testa o pior caso para algumas versões de QuickSort.
- **Vetor Quase Ordenado**: Testa casos intermediários.
- **Vetor Aleatório**: Um cenário comum para a maioria das aplicações práticas.

Medição do Tempo

Utilizei a função `clock()` da biblioteca `<time.h>` para medir o tempo de execução de cada versão do QuickSort.

Exemplo de Medição de Tempo:

```
#include <time.h>

clock_t inicio, fim;
double tempoTotal;

inicio = clock();
// Chame aqui a função QuickSort que deseja testar
fim = clock();

tempoTotal = ((double) (fim - inicio)) / CLOCKS_PER_SEC;
printf("Tempo de execução: %f segundos\n", tempoTotal);
```

Resultados

Após rodar os testes, os tempos de execução variaram de acordo com a estratégia de escolha do pivô. A mediana de três geralmente obteve o melhor desempenho, especialmente em casos onde o vetor estava quase ordenado, enquanto o uso do primeiro ou último elemento como pivô resultou em tempos de execução significativamente maiores para vetores ordenados.

Discussão

A escolha do pivô afeta diretamente o desempenho do QuickSort. Estratégias como a mediana de três elementos ajudam a evitar os piores casos, enquanto o uso do primeiro ou último elemento como pivô pode ser ineficiente em vetores já ordenados. O pivô aleatório oferece uma alternativa balanceada, mas a mediana de três geralmente oferece o melhor desempenho geral.