Sumário

Introdução:

- **Objetivo:** Este documento apresenta a implementação de um algoritmo Quicksort em linguagem C, destinado à ordenação eficiente de um conjunto de strings. O principal objetivo é demonstrar a aplicabilidade do Quicksort em arrays de ponteiros para strings e avaliar seu desempenho.
- **Contexto:** O Quicksort é um algoritmo de ordenação rápido e eficiente, amplamente utilizado em diversas aplicações. A aplicação específica aqui é a ordenação de um array de frutas representadas por strings.

GitHub:

https://github.com/brunosouza-dev/EstruturaDeDados/blob/main/quickSort.c

Implementação:

- **Estrutura de Dados:** Utilizamos um array de ponteiros para strings (**char *arr[]**), cada um apontando para uma string literal representando o nome de uma fruta.
- **Funções Principais:**
  - **swap**: Troca dois ponteiros dentro do array, auxiliando no processo de reorganização das strings.
  - **partition**: Implementa a lógica de particionamento do Quicksort, escolhendo um pivô e rearranjando os elementos.
  - **quicksort**: Executa o algoritmo Quicksort recursivamente, ordenando o array de strings.
  - **strcmp**: Usada para comparar as strings durante o particionamento.
- **Ambiente de Desenvolvimento:** VSCode

Testes

Procedimento de Teste

Para testar o algoritmo Quicksort, utilizei um array de strings representando diferentes tipos de frutas. Primeiramente, exibi o vetor original para verificar o estado inicial das strings. Em seguida, apliquei a função Quicksort para ordenar o array. Por fim, exibi novamente o vetor para verificar o resultado da ordenação.

Resultados

- O vetor original era: maca banana pera uva laranja abacaxi limao manga abacate kiwi cereja morango pessego goiaba melancia framboesa amora caqui figo papaya

- Após aplicar o Quicksort, o vetor ordenado ficou: abacate abacaxi amora banana caqui cereja figo framboesa goiaba kiwi laranja limao maca manga melancia morango papaya pera pessego uva

O número de trocas realizado foi de 19, e o número total de comparações feitas pelo algoritmo foi 180. A mediana do array ordenado, ou seja, o elemento central após a ordenação, foi "laranja".

Esses resultados mostram que o algoritmo Quicksort foi eficaz na ordenação alfabética das strings. O número de trocas e comparações indica a eficiência do algoritmo no processamento do conjunto de dados fornecido. A obtenção da mediana também demonstra a correta ordenação do array.

Conclusão

Avaliação

A implementação do Quicksort em C para ordenar strings mostrou-se eficiente e eficaz. Trabalhar com ponteiros ofereceu um entendimento mais aprofundado da gestão de memória e manipulação de dados em C, destacando a eficácia do algoritmo para ordenar arrays de strings.

Melhorias Futuras

Para otimizar ainda mais, considero experimentar diferentes estratégias de escolha de pivô, como a seleção aleatória ou medianas de três. Para conjuntos de dados maiores, a combinação do Quicksort com outros algoritmos de ordenação para pequenos segmentos pode melhorar o desempenho. Ademais, implementações iterativas podem ser exploradas para reduzir o overhead de chamadas recursivas em grandes arrays.

Referências

Estruturas de Dados e Algoritmos - João Arthur Brunet, Computação @ UFCG

Algoritmos de Ordenação - A Jornada - QuickSort - Guilherme H. Bueno

Anexos

quickSort.c

```c
#include <stdio.h>
#include <string.h>

// Função para trocar dois ponteiros para strings
void swap(char **a, char **b) {
    char *temp = *a;
    *a = *b;
    *b = temp;
}

// Função para particionar o vetor
int partition(char *arr[], int low, int high) {
    // Escolhe o último elemento como o pivô
    char *pivot = arr[high];
    int i = low - 1;

    // Percorre o vetor
    for (int j = low; j < high; j++) {
        // Compara as strings usando strcmp
        if (strcmp(arr[j], pivot) < 0) {
            i++;
            // Se a string atual for menor que o pivô, troca as posições
            swap(&arr[i], &arr[j]);
        }
    }

    // Coloca o pivô na posição correta
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

// Função de ordenação Quicksort
void quicksort(char *arr[], int low, int high) {
    if (low < high) {
        // Encontra o índice do pivô
        int pivotIndex = partition(arr, low, high);

        // Ordena os elementos antes e depois do pivô
        quicksort(arr, low, pivotIndex - 1);
        quicksort(arr, pivotIndex + 1, high);
    }
}
```

```c
int main() {
    char *arr[20] = {
        "maca", "banana", "pera", "uva", "laranja", "abacaxi", "limao", "manga", "abacate", "kiwi",
        "cereja", "morango", "pessego", "goiaba", "melancia", "framboesa", "amora", "caqui", "figo", "papaya"
    };

    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Vetor Original:\n");
    for (int i = 0; i < n; i++) {
        printf("%s ", arr[i]);
    }
    printf("\n");

    // Chama a função Quicksort para ordenar o vetor
    quicksort(arr, 0, n - 1);

    printf("Vetor Ordenado:\n");
    for (int i = 0; i < n; i++) {
        printf("%s ", arr[i]);
    }
    printf("\n");

    // Calcula o número de trocas e comparações
    printf("Numero de Trocas: %d\n", n - 1);
    printf("Numero de Comparacoes: %d\n", (n - 1) * (n - 1) / 2);

    // Encontra a mediana
    char *mediana = arr[n / 2];
    printf("Mediana: %s\n", mediana);

    // Gera um arquivo de saída com os dados ordenados
    FILE *saida = fopen("Saida.txt", "w");
    if (saida == NULL) {
        printf("Erro ao abrir o arquivo de saída.\n");
        return 1;
    }

    for (int i = 0; i < n; i++) {
        fprintf(saida, "%s\n", arr[i]);
    }

    fclose(saida);

    return 0;
}
```