Trabalho Prático III: Dominando Heaps em C

Introdução

Este trabalho prático tem como objetivo aprofundar seu conhecimento e habilidades na manipulação de estruturas de dados **Heap Máxima (Max Heap)** e **Heap Mínima (Min Heap)** em C. Você irá trabalhar sobre um código-base existente, implementando novas funcionalidades e modificando as existentes para entender melhor o comportamento e as aplicações dessas estruturas.

Código Base Fornecido

O código fornecido já inclui as funções básicas para:

- `trocar`: Troca dois elementos.
- `obterPai`, `obterFilhoEsquerdo`, `obterFilhoDireito`: Funções auxiliares para navegação na heap.
- `imprimirVetor`: Para visualizar o array.
- `maxHeapificarParaBaixo` / `minHeapificarParaBaixo`: Mantém a propriedade de heap após uma modificação em um nó.
- `construirMaxHeap` / `construirMinHeap`: Constrói uma heap a partir de um array desordenado.
- `main`: Com exemplos de construção de ambas as heaps.

Desafios e Modificações Propostas

Sua tarefa será estender e modificar o código base, implementando as seguintes funcionalidades:

Desafio 1: Inserção de Elemento em Heap (Max e Min)

Atualmente, o código apenas constrói uma heap a partir de um vetor inicial. Adicionar um novo elemento a uma heap existente requer uma operação diferente: **heapificar para cima**.

- **Objetivo:** Implementar funções para inserir um novo valor mantendo as propriedades da heap.

1. **`maxHeapificarParaCima(int arr[], int i)`:**
    - Crie esta função. Ela deve receber o array `arr` e o índice `i` do novo elemento inserido (que está na "folha" ou em uma posição temporária).
    - Ela deve comparar o elemento em `i` com seu pai (`obterPai(i)`). Se o elemento filho for maior que o pai (para Max Heap), troque-os e continue o

processo recursivamente ou em um loop até que o elemento esteja na posição correta ou se torne a raiz.
    - **Dica:** Utilize a função `obterPai` e `trocar` já existentes.
2. **`inserirMaxHeap(int arr[], int *tamanho, int valor)`:**
    - Crie esta função. Ela deve receber o array `arr`, um ponteiro para o `tamanho` atual da heap (pois ele irá aumentar), e o `valor` a ser inserido.
    - Primeiro, incremente o `tamanho` do array.
    - Adicione o `valor` na **última posição** do array (o novo `tamanho - 1`).
    - Chame `maxHeapificarParaCima` para posicionar o `valor` corretamente na Max Heap.
3. **`minHeapificarParaCima(int arr[], int i)`:**
    - Similar a `maxHeapificarParaCima`, mas para Min Heap. Se o filho for **menor** que o pai, troque-os e continue.
4. **`inserirMinHeap(int arr[], int *tamanho, int valor)`:**
    - Similar a `inserirMaxHeap`, mas chamando `minHeapificarParaCima`.

- **Testes no `main`:**
    - Após construir uma Max Heap (ou Min Heap) no `main`, insira 2-3 novos elementos (ex: `15`, `0`, `7` para Max Heap; `0`, `2`, `10` para Min Heap) e imprima o vetor após cada inserção para verificar se a propriedade de heap foi mantida.

Desafio 2: Extração de Elemento (Remoção da Raiz) em Heap (Max e Min)

A operação mais comum em heaps é a extração do elemento de maior prioridade (a raiz).

- **Objetivo:** Implementar funções para remover o elemento da raiz (o maior em Max Heap, o menor em Min Heap) e manter a propriedade de heap.

1. **`extrairMaxHeap(int arr[], int *tamanho)`:**
    - Crie esta função. Ela deve retornar o valor do elemento de maior prioridade (a raiz).
    - Se a heap estiver vazia, retorne um valor sentinela (ex: `-1`) ou trate o erro.
    - Armazene o valor da raiz (`arr[0]`).
    - Mova o **último elemento** da heap (`arr[*tamanho - 1]`) para a posição da raiz (`arr[0]`).
    - **Decrementa** o `tamanho` da heap.
    - Chame `maxHeapificarParaBaixo(arr, *tamanho, 0)` para restaurar a propriedade da Max Heap a partir da nova raiz.
    - Retorne o valor armazenado da raiz original.
2. **`extrairMinHeap(int arr[], int *tamanho)`:**

    - Similar a `extrairMaxHeap`, mas para Min Heap, chamando `minHeapificarParaBaixo`.
- **Testes no `main`:**
    - Após as inserções (Desafio 1), extraia 2-3 elementos de cada heap (Max e Min).
    - Imprima o valor extraído e o estado do vetor após cada extração para verificar o funcionamento.

Desafio 3: Implementando Heap Sort (Ordenação com Heap)

Um dos usos mais poderosos de heaps é o algoritmo de ordenação Heap Sort. Ele utiliza as operações de construção e extração.

- **Objetivo:** Implementar o algoritmo Heap Sort.

1. **`heapSort(int arr[], int tamanho)`:**
    - Crie esta função. Ela deve ordenar o array `arr` em ordem crescente.
    - **Passo 1: Construir uma Max Heap:** Chame `construirMaxHeap(arr, tamanho)` no início para transformar o array em uma Max Heap.
    - **Passo 2: Extração e Reconstrução:**
        - Itere de `tamanho - 1` até `1` (ou seja, do último elemento até o segundo).
        - Em cada iteração, troque a **raiz da heap (`arr[0]`)** com o elemento na **posição atual do loop (`arr[i]`)**.
        - Considere que a parte do array que está sendo ordenada diminui a cada passo. Chame `maxHeapificarParaBaixo(arr, i, 0)` para restaurar a propriedade da Max Heap nos elementos restantes (excluindo os que já foram "ordenados" no final do array).
- **Testes no `main`:**
    - Crie um **novo vetor desordenado** (ex: `{7, 2, 9, 1, 5, 3, 8, 4, 6}`).
    - Imprima o vetor original.
    - Chame `heapSort` para ordená-lo.
    - Imprima o vetor ordenado.

Desafio 4 (Opcional - Mais Avançado): Heap Genérica ou Heapify Melhorado

Se você já domina os desafios anteriores, explore estas opções:

1. **Heap Genérica (Usando `void*` e Ponteiros para Funções):**

    - (Mais complexo) Modifique as funções de heap (`maxHeapificarParaBaixo`, `minHeapificarParaBaixo`, etc.) para aceitarem um array de tipo `void*` e um ponteiro para uma função de

comparação (que receba dois `void*` e retorne um `int` indicando a relação, como `strcmp` ou uma função personalizada). Isso permitiria que sua heap armazenasse e ordenasse qualquer tipo de dado (structs, floats, etc.).

2. **`heapify` Iterativo (Sem Recursão):**
    - (Desafio de otimização) Altere as implementações de `maxHeapificarParaBaixo` e `minHeapificarParaBaixo` para que usem um loop `while` em vez de chamadas recursivas. Isso pode ser ligeiramente mais eficiente em algumas situações, evitando a sobrecarga de chamadas de função recursivas.

Requisitos de Entrega

1. Um único arquivo `.c` contendo todas as funções auxiliares, as implementações de Max/Min Heap (incluindo as novas funções de inserção, extração e Heap Sort) e uma função `main` bem organizada com testes claros para cada desafio.
2. Adicione **comentários** no código explicando cada função e as principais partes da lógica.
3. Garanta que o código **compile sem erros** e **execute corretamente** exibindo os resultados esperados no console.

Este trabalho prático cobrirá as operações fundamentais de heaps e aplicará seu conhecimento de C em um contexto prático de estrutura de dados. Boa sorte!