Guia de Estudos
Práticas com EDs

Lucas Vinícius de Lima Assis
(62) 9 9973-7345

1 de dezembro de 2023

Considerações iniciais

"Este é um pequeno guia sobre prática com algumas estruturas de dados. Como o último Guia de Estudos foi sobre os pré-requisitos para se trabalhar com estruturas de dados (Ponteiros, malloc e structs), neste guia eu trago algumas plataformas que vocês podem usar para encontrar questões, sempre procurando dar a vocês os materiais para terem independência no seu aprendizado.

Já adianto, recomendo que implementem, no mínimo uma vez, uma lista encadeada, duplamente encadeada, circular, tratem como uma pilha, e depois como uma fila, e façam no mínimo as operações básicas: adicionar no começo, no final, printar todos os elementos, busca de elementos. Essas são operações básicas.

Parece muito, mas vocês vão ver que é tranquilo, exemplo: entre uma pilha e uma fila, a única diferença é a inserção e retirada, a busca é igual à da lista encadeada, não precisa repetir.

Fazer as operações ao menos uma vez ativa teu cérebro para essa nova maneira de pensar usando ponteiros."

-O Monitor

Beecrowd

O Beecrowd é um site de programação competitiva. Ele contém um banco de questões enorme. A melhor parte é que o próprio site verifica se sua resposta está correta, passando seu código por centenas de casos de testes.

Sugiro então que criem uma conta *(já aviso que é chatinho, mas tenham animo)* e vão na aba Problems→Structs que vão encontrar problemas sobre estruturas de dados. Aqui estão 4 problemas que separei:

- Parenthesis Balance I
- Diamonds and Sand
- Infix Postfix
- LEXSIM - Sintatic and Lexical Avaliator

Uma vantagem desse tipo de questão é que as perguntas são abertas, vocês podem resolver de diversas maneiras, e vocês que devem avaliar qual estrutura de dados utilizar, ou até mesmo, se realmente precisam utilizar alguma.

Pessoalmente, quando eu resolvo os exercícios no beecrowd, muitas vezes eu opto por implementações mais simples das estruturas de dados, porque eles me dão um problema pontual, e não um software que pode expandir e, então, requer o código mais genérico possível. Assim, recomendo que se resolverem de uma maneira mais, digamos, *'sagaz'*, se desafiem a resolver com uma implementação clássica, e ver a diferença de performance entre as duas soluções.

Sugestões Materiais Externos

Sugiro que procurem também materiais pela internet mais específicos sobre o conteúdo. O Instituto de Computação (IC) da Unicamp possui diversos materiais sobre o assunto, estudei por lá a parte de listas encadeadas. A seguinte questão foi retirada do Material da professora Thelma Chiossi.

Subdividir uma *Linked List*

Considere a implementação:

```
typedef struct lista{
    int info;
    struct lista *prox;
}Lista;
```

Escreva uma função que receba, como parâmetros, uma lista e um número inteiro n, e divida a lista em duas, de forma que a segunda lista comece no primeiro nó logo após a ocorrência de n na lista original. A função deve retornar um ponteiro para a segunda subdivisão da lista original, enquanto L (cabeça da lista original) deve continuar apontando para o primeiro elemento da primeira subdivisão da lista.

Depois deem uma olhada e tentem fazer as outras questões, vai que você encontra algo que parece difícil lá.

Dica: Aprendam a fazer bibliotecas

Sendo bem direto, é muito útil ter uma biblioteca com uma implementação de uma lista encadeada, outra com uma lista duplamente encadeada, outra com uma pilha, uma pilha com implementação de vetor, com implementação de lista encadeada, e por aí vai... Vocês vão precisar tantas vezes dessas mesmas implementações, que vale a pena guardar tudo num arquivo, e quando precisar, só dar um `#include "BolsaDaHermione.h"`. Sugiro esse material do IC da Unicamp.