UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS CRATEÚS
SISTEMAS DE INFORMAÇÃO
DOCENTE: RAFAEL MARTINS BARROS
DISCIPLINA: ESTRUTURA DE DADOS

DISCENTES: ANA JAZA TAVARES SOARES - 558540, ANA LETICIA DE OLIVEIRA SOARES - 555346.

RELATÓRIO: LISTA DUPLAMENTE ENCADEADA

CRATEÚS
2024

SUMÁRIO

CRATEÚS
2024

1. Objetivo

O objetivo deste trabalho foi implementar uma lista duplamente encadeada em linguagem C, aplicando os conceitos de estrutura de dados. O projeto foi desenvolvido em dupla por Jaza e Ana Letícia, utilizando uma abordagem incremental, com versionamento por meio de commits no GitHub.

A lista duplamente encadeada possui como característica nós que possuem um ponteiro para o nó anterior e outro ponteiro para o próximo nó, permitindo a navegação bidirecional pela lista. A implementação foi feita utilizando duas structs: uma que representa os nós da lista e outra para a própria lista. A Struct No contém um valor inteiro e dois ponteiros: um para o nó anterior e outro para o próximo nó. Já a Struct Lista possui um ponteiro para o início da lista, um para o fim da lista e uma variável que armazena o tamanho atual.

Diferentemente de uma lista simplesmente encadeada, onde cada nó possui apenas um ponteiro para o próximo nó, a lista duplamente encadeada possui ponteiros tanto para o próximo quanto para o nó anterior - o que oferece maior flexibilidade, pois permite a navegação bidirecional, facilitando operações como remoção e inserção em ambas as extremidades da lista. Sobre termos de uso de memória, a lista duplamente encadeada vai requerer mais espaço, pois cada nó armazena dois ponteiros, enquanto a lista simplesmente encadeada armazena apenas um. A lista duplamente encadeada é mais eficiente em certas operações, como remoção de nós intermediários, uma vez que não é necessário percorrer a lista desde o início para encontrar o nó anterior ao nó alvo, como ocorre na lista simplesmente encadeada.

2. Principais funções implementadas e Big-O

2.1. Criar Lista

Inicializa uma nova lista vazia.

O(1) – A lista é criada e inicializada com os ponteiros de início e fim apontando para NULL e o tamanho inicial 0.

2.2. Inserir Elemento:

Insere um elemento inteiro na lista, mantendo a ordem crescente.

O(n) – No pior caso, é necessário percorrer toda a lista até encontrar a posição correta para inserção, comparando os valores.

2.3. Remover Elemento por Valor:

Remove a primeira ocorrência de um elemento com um valor específico.

O(n) – É necessário percorrer a lista até encontrar o valor especificado.

2.4. Obter Elemento por Posição:

Retorna o elemento na posição especificada.

O(n) – No pior caso, a lista precisa ser percorrida até a última posição.

2.5. Procurar Elemento:

Retorna a posição da primeira ocorrência de um valor especificado. Se não encontrado, retorna uma indicação de não encontrado.

O(n) – No pior caso, o valor pode estar no final da lista ou não estar presente, exigindo a verificação de todos os nós.

2.6. Tamanho da Lista:

Retorna o número de elementos atualmente na lista.

O(1) – Apenas retorna o valor armazenado na variável que mantém o tamanho.

2.7. Imprimir Lista:

Exibe todos os elementos da lista em ordem crescente.

O(n) – No pior caso a função percorre toda a lista para exibir os valores.

2.8. Imprimir Lista Reversa:

Exibe todos os elementos da lista em ordem decrescente.

O(n) – Semelhante à impressão em ordem crescente, porém começando do final da lista e retrocedendo percorrendo os ponteiros de volta.

3. Como o trabalho foi organizado

Para a realização deste trabalho, formamos um grupo na plataforma de mensagens WhatsApp focado em discutir e buscar soluções para cada tarefa solicitada. Com o objetivo de manter uma boa organização, divisão de responsabilidades e divisão das implementações. Jaza ficou responsável pelas quatro primeiras funções, enquanto Ana Letícia implementou as demais funções, além da parte de criação do menu de interação com o usuário.

4. Principais dificuldades

Uma das principais dificuldades encontradas foi garantir que as operações de

inserção e remoção mantivessem a integridade dos ponteiros de cada nó, especialmente na manutenção da ordem crescente e na remoção de nós. Testar e verificar o comportamento dos ponteiros foi um dos maiores desafios, principalmente em cenários onde os nós eram inseridos ou removidos de posições intermediárias.

5. Conclusão

Concluímos que, apesar dos desafios enfrentados, o processo de implementação da lista duplamente encadeada foi uma experiência enriquecedora. Através de uma abordagem rigorosa de testes e validações, conseguimos identificar e corrigir os problemas, o que nos proporcionou uma compreensão mais aprofundada do funcionamento de estruturas de dados dinâmicas. Essa experiência reforçou os conceitos teóricos aprendidos em sala de aula e nos preparou para desenvolver soluções mais complexas no futuro.