GOVERNO DO ESTADO DO PIAUÍ UNIVERSIDADE ESTADUAL DO PIAUÍ
CAMPUS PROFESSOR ANTÔNIO GIOVANNI ALVES DE SOUSA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: Estrutura De Dados
ALUNO: Samuel Sousa Lustosa

Introdução

https://github.com/sael69/lista_duplamente-encadeada-ED.git
A estrutura de dados é uma parte fundamental da programação que influencia diretamente o desempenho e eficiência de algoritmos. A lista duplamente encadeada é uma estrutura versátil que oferece vantagens específicas em comparação com outras estruturas de dados. Sua capacidade de percorrer a lista nos dois sentidos e realizar operações de inserção e remoção eficientes a torna uma escolha valiosa para várias aplicações.

Justificativa da Escolha da Lista Duplamente Encadeada

A decisão de usar a lista duplamente encadeada para este projeto foi motivada por suas características distintas. A habilidade de percorrer a lista nos dois sentidos, combinada com operações eficientes de inserção e remoção, proporciona versatilidade e desempenho. Essa escolha é especialmente valiosa em situações em que a necessidade de acessar elementos próximos, tanto para frente quanto para trás na lista, é um requisito essencial.

Implementação

Estrutura de dados:

```
5  struct Node {
6      int data;
7      struct Node* prev;
8      struct Node* next;
9  };
10
11 struct DoublyLinkedList {
12     struct Node* head;
13 };
```

Funções Implementadas

createNode
- Descrição: Aloca memória para um novo nó e inicializa seus campos.
- Parâmetros: Valor a ser armazenado no nó.
- Retorno: Ponteiro para o novo nó.

insertEnd
- Descrição: Insere um novo nó no final da lista.
- Parâmetros: Lista e valor a ser inserido.

removeNode
- Descrição: Remove um nó específico da lista.
- Parâmetros: Lista e valor a ser removido.

search
- Descrição: Procura por um valor na lista.
- Parâmetros: Lista e valor a ser encontrado.
- Retorno: Ponteiro para o nó encontrado ou NULL se não encontrado.

printList
- Descrição: Imprime os elementos da lista.

freeList
- Descrição: Libera a memória alocada para todos os nós da lista.

createList
- Descrição: Redefine a cabeça da lista como NULL, criando uma nova lista vazia.

Resultados de Desempenho

O experimento envolveu a realização de 100,000 operações de inserção para avaliar o desempenho da lista duplamente encadeada.

- **Tempo de Inserção:**
  - Resultado: 7.162494 segundos para 100,000 operações de inserção.
  - Comentário: O tempo de inserção é considerado razoável, especialmente considerando a natureza linear da inserção no final de uma lista duplamente encadeada. A variação desse tempo pode depender da implementação específica, do hardware e do ambiente de execução.
- **Tempo de Remoção:**
  - Resultado: 0.000249 segundos para 100,000 operações de remoção.
  - Comentário: O tempo de remoção é notavelmente rápido. Pode haver otimizações no compilador ou outros fatores que explicam essa eficiência. Além disso, a remoção do elemento do meio

pode ser uma operação mais eficiente do que a média.

- **Tempo de Criação:**
  - Resultado: 0.001600 segundos para a operação de criação.
  - Comentário: A função de criação (createList) é responsável por redefinir a cabeça da lista como NULL, criando efetivamente uma nova lista vazia. O tempo de criação é pequeno, conforme esperado, dada a simplicidade dessa operação.

Observações Gerais:

- Os resultados estão em linha com as expectativas teóricas, levando em consideração as características específicas das operações realizadas.
- Fatores como cache do processador e otimizações do compilador podem influenciar os resultados.

Esses resultados fornecem insights valiosos sobre o desempenho da implementação da lista duplamente encadeada em seu ambiente de execução. Para análises mais detalhadas ou otimizações, considere aspectos específicos da implementação e das operações realizadas.

Análise Crítica

Desempenho das Operações

- Inserção e Remoção:
  - Adicionar e remover itens no final da lista é rápido.
  - Remover um item específico pode levar mais tempo, especialmente se estiver longe do final.
- Busca:
  - Encontrar um item requer percorrer a lista, o que pode demorar se a lista for grande.

Complexidade Temporal e Espacial

- Tempo:
  - Adicionar e remover são operações rápidas, mas encontrar itens pode ser mais lento em comparação com outras estruturas.
  - O desempenho geral depende das operações mais frequentes na sua aplicação.
- Espaço:
  - A lista usa um pouco mais de espaço para manter os links entre os itens, mas isso não é geralmente um grande problema, a menos que a memória seja muito limitada.

Comparação com Outras Estruturas de Dados

- A lista duplamente encadeada é boa para adicionar e remover itens no final.
- Para buscar itens de forma mais eficiente, outras estruturas como árvores podem ser mais adequadas.

Cenários de Utilização Adequada

- Quando é necessário acessar elementos na lista tanto para frente quanto para trás.
- Operações frequentes de inserção e remoção no final da lista.

Código Fonte:

https://github.com/sael69/lista_duplamente-encadeada-ED.git