UNIVERSIDADE FEDERAL DE MATO GROSSO
CAMPUS UNIVERSITÁRIO DO ARAGUAIA
INSTITUTO DE CIÊNCIAS EXATAS E DA TERRA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Fernanda Lima de Souza
Pedro Henrique Camelo de França

RELATÓRIO DE ESTRUTURA DE DADOS I
Análise das Listas Estática, Encadeada e Duplamentente Encadeada

Barra do Garças - MT
2025

O código tem como objetivo medir o tempo de execução da inserção de elementos em três tipos de listas: **estática, encadeada e duplamente encadeada**. Ele faz isso lendo arquivos contendo números inteiros aleatórios e inserindo esses valores nas listas utilizando diferentes estratégias.

Estrutura	Qtd.Inico	Qtd.Fim	Qtd.Posição	Menor Tempo	Maior Tempo	Média
Lista Estatica	3343	3340	3317	0.039784 segundos	0.068792 segundos	0.054630 segundos
Lista Encadeada	3280	3398	3322	0.055171 segundos	0.078279 segundos	0.065164 segundos
Lista Duplamente Encadeada	3363	3403	3234	0.044342 segundos	0.088599 segundos	0.066090 segundos

Avaliação da Ocupação de Memória nas Estruturas de Lista

É possível avaliar a ocupação de memória em diferentes estruturas de lista: estática, encadeada e duplamente encadeada.

- **Lista Estática**: A memória é alocada de forma contígua e fixa durante a compilação. Sua ocupação é determinada pelo número máximo de elementos multiplicado pelo tamanho de cada elemento.
- **Lista Encadeada**: A alocação de memória é dinâmica. Cada nó contém um ponteiro para o próximo nó, além dos dados armazenados, o que aumenta a ocupação de memória.
- **Lista Duplamente Encadeada**: Similar à lista encadeada simples, porém, cada nó possui um ponteiro adicional para o nó anterior, aumentando ainda mais a ocupação de memória.

Eficiência em Uso de Memória

A lista estática é a mais eficiente em termos de memória, pois evita o uso de ponteiros. Em seguida, vem a lista encadeada, seguida pela lista duplamente encadeada, que requer mais memória devido ao ponteiro extra em cada nó.

Análise do Código para Medição de Tempo de Inserção

O código desenvolvido tem o objetivo de medir o tempo de execução da inserção de elementos em diferentes tipos de lista, utilizando arquivos contendo números inteiros.

Lista Estática

1. **Inicialização**: A lista é inicializada com um contador igual a zero.
2. **Métodos de Inserção**:
    - `insereListaFim()`: Adiciona um número ao final da lista.
    - `insereListaInicio()`: Adiciona um número no início, respeitando um limite de inserções.
    - `insereListaPosicao()`: Insere um número em uma posição aleatória, respeitando um limite.
3. **Leitura de Arquivo**: A função `ler_arquivo_e_inserir()` lê números de um arquivo e os insere aleatoriamente em uma das três estratégias.
4. **Medição de Tempo**: `medir_tempo_execucao()` usa `clock()` para calcular o tempo de inserção.
5. **Execução**: A função `main()` executa o teste para 10 arquivos e calcula o tempo médio.

Lista Encadeada

1. **Inicialização**: A lista é iniciada com `inicio` e `fim` como `NULL` e um contador zerado.
2. **Métodos de Inserção**:
    - `insereListaFim()`: Insere um número no final da lista.
    - `insereListaInicio()`: Insere um número no início da lista.
    - `insereListaPosicao()`: Insere um número em uma posição aleatória.
3. **Leitura de Arquivo**: `ler_arquivo_e_inserir()` lê números de um arquivo e insere na lista com uma estratégia aleatória.
4. **Liberação de Memória**: `limpaLista()` percorre e libera a memória dos nós.
5. **Medição de Tempo**: `medir_tempo_execucao()` usa `clock()` para medir o tempo de inserção.
6. **Execução**: `main()` executa os testes para 10 arquivos e calcula a média do tempo de execução.

Lista Duplamente Encadeada

1. **Inicialização**: Ponteiros *`inicio`* e `fim` começam como `NULL` e o contador é zerado.
2. **Métodos de Inserção**:
    - `insereListaFim()`: Insere um número no final da lista.
    - `insereListaInicio()`: Insere um número no início da lista.
    - `insereListaPosicao()`: Insere um número em uma posição aleatória.

3. **Leitura de Arquivo**: `ler_arquivo_e_inserir()` lê um arquivo e insere os valores aleatoriamente.
4. **Liberação de Memória**: `limpaLista()` libera os nós para evitar vazamento de memória.
5. **Medição de Tempo**: `medir_tempo_execucao()` usa `clock()` para medir a inserção.
6. **Execução**: `main()` executa os testes para 10 arquivos e calcula a média do tempo de execução.

Conclusão: Estrutura Mais Eficiente

Após os testes, a **lista estática** apresentou o melhor desempenho em tempo de execução, sendo a mais rápida entre as três abordagens. Isso ocorre porque a alocação de memória é feita de forma contígua e não há sobrecarga de ponteiros, tornando as operações mais rápidas.

Já as **listas encadeadas** e **duplamente encadeadas** possuem maior flexibilidade, permitindo inserções dinâmicas sem desperdício de espaço. No entanto, o uso de ponteiros impacta o tempo de execução, tornando-as menos eficientes do que a lista estática em termos de velocidade.