Projeto de Biblioteca de Classes para Manipulação de Vetor Dinâmico em C++

Curso: TADS
Matéria: Algoritmos
Aluno: João Pedro Dantas Magalhães
Link GitHub:https://github.com/joaodantass15/vetor_dinamico_c-.git

01 - Introdução

Este trabalho tem como objetivo a implementação de uma biblioteca de classes em C++ para manipulação de um vetor dinâmico de números inteiros. A biblioteca contará com duas classes distintas: uma utilizando alocação dinâmica de arrays e outra usando lista duplamente ligada. As ferramentas utilizadas foram: GitHub, VsCode, Git, g++ no windows.

02 - Vetores Dinâmicos

Um vetor dinâmico é uma estrutura de dados que ajusta sua capacidade de armazenamento conforme a quantidade de elementos inseridos ou removidos. Nesse projeto, utilizaremos lista ligada e alocação dinâmica, separando-os em classes e com os métodos fazer os testes.

Na classe “linked_list” para realizar os testes com lista ligada o compilador não consegue encontrar as definições das funções da classe, o que resulta em "undefined reference". Assim, tive que fazer alterações no código da classe. Dessa forma, foi possível compilar utilizando o editor VsCode.

03 - Implementação

03.01 - Organização dos arquivos fontes

Os arquivos fontes estão organizados da seguinte maneira:

- `array_list.hpp`: Implementação da classe `array_list` usando alocação dinâmica de arrays.
- `linked_list.hpp`: Implementação da classe `linked_list` usando lista duplamente ligada.
- `main.cpp`: Arquivo principal para execução e testes das implementações.
- `README.md`: Instruções para compilação e execução dos testes.
- Casos de teste: Arquivos de testes específicos para as implementações.

03.02 - Arrays com Alocação Dinâmica

- Arrays com alocação dinâmica são estruturas de dados em que se pode criar um vetor com um tamanho que pode ser definido em tempo de execução, alocação de memória durante a execução do programa, em vez de durante a compilação.

- **Cabeçalho (`array_list.hpp`)**

03.03 - Lista Ligada

- Cada elemento, o nó, contém um valor e uma referência/ponteiro para o próximo nó na sequência. Essa estrutura permite uma inserção e remoção eficiente de elementos, já que não requer a movimentação de outros elementos.
- **Cabeçalho (`linked_list.hpp`)**

03.04 - Testes

Os testes para para implementar e testar os algoritmos foram feitos baseados nas seguintes funções/especificações:

- Inserção consecutiva de elementos no início do vetor.
- Inserção consecutiva de elementos no final do vetor.
- Remoção de elementos pelo índice.
- Remoção de elementos no início do vetor.
- Remoção de elementos no final do vetor.

Para cada classe, ou seja, com alocação dinâmica e lista duplamente encadeadas, foram feitos os 5 testes e observamos a complexidade do algoritmo e tempos de execução com a entrada 10.000.

É importante salientar que cada teste foi implementado no VsCode, gerando um tempo de execução e analisando o código podemos calcular a complexidade do algoritmo utilizando notação big-OH. Com essas informações, desenvolvi a tabela a seguir (tópico 04).

04 - Resultados

04.01 - Testes: Alocação dinâmica:(no tempo, foram testadas com duas formas diferentes de escrever as classes, mas o código de testes era o mesmo, de fato, fiquei com dúvida nessa parte).

TESTE	ENTRADA	TEMPO	BIG-OH
Inserção consecutiva de elementos no início do vetor.	10.000	0 ms / 199.885 ms	O(n^2)
Inserção consecutiva de elementos no final do vetor.	10.000	0 ms / 1.001 ms	O(n)

Conjunto de remoções de elementos pelo índice.	10.000	0 ms / 225.889 ms	O(n^2)
Conjunto de remoções de elementos no início do vetor.	10.000	0 ms / 218.876 ms	O(n^2)
Conjunto de remoções de elementos no final do vetor.	10.000	0 ms	O(n)

04.02 - Testes: Listas duplamente encadeadas:

TESTE	ENTRADA	TEMPO	BIG-OH
Inserção consecutiva de elementos no início do vetor.	10.000	0.008993 seg	O(n)
Inserção consecutiva de elementos no final do vetor.	10.000	0.008976 seg	O(n) / O(1)
Conjunto de remoções de elementos pelo índice.	10.000	0 seg	O(n^2)
Conjunto de remoções de	10.000	0.565676 seg	O(n) / O(1)

elementos no início do vetor.			
Conjunto de remoções de elementos no final do vetor.	10.000	0.560696 seg	O(n)

Resultados da Análise de Complexidade

- **Array Dinâmico**:
  - **Inserção no Final (`push_back`)**: O tempo de execução médio é O(1), mas O(n) no pior caso quando é necessário aumentar a capacidade.
  - **Inserção em Índice Específico (`insert_at`)**: O tempo de execução é O(n) devido ao deslocamento dos elementos.
  - **Remoção em Índice Específico (`remove_at`)**: O tempo de execução também é O(n) pela necessidade de deslocar elementos.

- **Lista Duplamente Ligada**:
  - **Inserção no Final (`push_back`)**: O tempo de execução é O(1) se o ponteiro para o último elemento for mantido.
  - **Inserção em Índice Específico (`insert_at`)**: O tempo de execução é O(n) devido à necessidade de percorrer a lista para encontrar a posição.
  - **Remoção em Índice Específico (`remove_at`)**: O tempo de execução é O(n) pelo mesmo motivo da inserção.

05 - Conclusão

Portanto, na classe com alocação dinâmica o menor tempo de execução foi no teste “Conjunto de remoções de elementos no final do vetor”. Já na classe com lista duplamente encadeada foi o teste “Conjunto de remoções de elementos pelo índice”.

Os requisitos específicos da aplicação devem decidir se usar um array dinâmico ou uma lista duplamente ligada. A lista duplamente ligada pode ser a melhor opção se o desempenho das operações de inserção e remoção for importante. Por outro lado, um array dinâmico pode ser mais adequado se for necessário acesso rápido aos elementos. Assim, os testes de desempenho realizados confirmaram as análises teóricas, demonstrando a eficiência de cada estrutura em diferentes cenários.